TransformerSum

Release 1.0.0

Hayden Housen

Oct 13, 2021

GENERAL:

TransformerSum

CHAPTER

ONE

ABOUT

1.1 Overview

TransformerSum is a library that aims to make it easy to *train*, *evaluate*, and *use* machine learning **transformer models** that perform **automatic summarization**. It features tight integration with huggingface/transformers which enables the easy usage of a **wide variety of architectures** and **pre-trained models**. There is a heavy emphasis on code **readability** and **interpretability** so that both beginners and experts can build new components. Both the extractive and abstractive model classes are written using pytorch_lightning, which handles the PyTorch training loop logic, enabling **easy usage of advanced features** such as 16-bit precision, multi-GPU training, and much more. TransformerSum supports both the extractive and abstractive summarization of **long sequences** (4,096 to 16,384 tokens) using the longformer (extractive) and LongformerEncoderDecoder (abstractive), which is a combination of BART (paper) and the longformer. TransformerSum also contains models that can run on resource-limited devices while still maintaining high levels of accuracy. Models are automatically evaluated with the **ROUGE metric** but human tests can be conducted by the user.

Check out the *installation instructions* and the *tutorial* to get started training models and summarizing text.

Both extractive and abstractive processed datasets and trained models can be found in their respective sections. Alternatively, you can browse from the root folder in Google Drive that contains all of the models and datasets: TransformerSum Root Folder.

1.2 Features

- For extractive summarization, compatible with every huggingface/transformers transformer encoder model.
- For abstractive summarization, compatible with every huggingface/transformers EncoderDecoder and Seq2Seq model.
- Currently, 10+ pre-trained extractive models available to summarize text trained on 3 datasets (CNN-DM, WikiHow, and ArXiv-PebMed).
- Contains pre-trained models that excel at summarization on resource-limited devices: On CNN-DM, `mobilebert-uncased-ext-sum` achieves about 97% of the performance of BertSum while containing 4.45 times fewer parameters. It achieves about 94% of the performance of MatchSum (Zhong et al., 2020), the current extractive state-of-the-art.
- Contains code to train models that excel at summarizing long sequences: The longformer (extractive) and LongformerEncoderDecoder (abstractive) can summarize sequences of lengths up to 4,096 tokens by default, but can be trained to summarize sequences of more than 16k tokens.
- Integration with huggingface/nlp means any summarization dataset in the `nlp` library can be used for both abstractive and extractive training.

- "Smart batching" (extractive) and trimming (abstractive) support to not perform unnecessary calculations (speeds up training).
- Use of `pytorch_lightning` for code readability.
- Extensive documentation.
- Three pooling modes (convert word vectors to sentence embeddings): mean or max of word embeddings in addition to the CLS token.

1.3 Significant People

The project was created by Hayden Housen during his junior and senior years of high school as part of the Science Research program. It is actively maintained and updated by him and the community. You can contribute at HHousen/TransformerSum.

1.4 Extractive vs Abstractive Summarization

Models that perform **extractive summarization** essentially pick the best most representative sentences and copy them into a summary. Models that perform **abstractive summarization** generate new sentences that capture general ideas.

Extractive summarization is a **binary classification problem**. Either classify the sentence as "should be in he summary" or "should NOT be in the summary".

Abstractive summarization is a **sequence to sequence text generation problem**. This is significantly more difficult than extractive summarization since the machine has to synthesize the information it "reads" into a new form.

1.5 ROUGE Scores

This project uses ROUGE to evaluate summarization quality. ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, is a set of metrics used to evaluate automatic summarization systems. However, automatic metrics, such as ROUGE and METEOR, have serious limitations. They only assess content selection by calculating lexical overlap and do not account for other quality aspects, such as fluency, grammaticality, or coherence. More information about the limitations of ROUGE in sebastianruder/NLP-progress.

Links:

- ROUGE Paper (PDF)
- ROUGE Score Wikipedia
- Overview of How ROUGE Works (Archive)

This project integrates with rouge-score and pyrouge and either can be used when calculating ROUGE scores during the testing phase.

`rouge-score` is the default option. It is a pure python implementation of ROUGE designed to replicate the results of the official ROUGE package. While this option is cleaner (no perl installation required, no temporary directories, faster processing) than using `pyrouge`, this option should not be used for official results due to minor score differences with `pyrouge`.

`pyrouge` is a python interface to the official ROUGE 1.5.5 perl script. Using this option will produce official scores, but it requires a complicated setup. To install ROUGE 1.5.5 I followed this StackOverflow answer and ran the below commands from Kavita Ganesan (Archive) to fix the WordNet exceptions:

```
cd data/WordNet-2.0-Exceptions/
./buildExeptionDB.pl . exc WordNet-2.0.exc.db

cd ../
rm WordNet-2.0.exc.db
ln -s WordNet-2.0-Exceptions/WordNet-2.0.exc.db WordNet-2.0.exc.db
```

Note: The official ROUGE website was http://www.berouge.com/Pages/default.aspx but has been offline for many years. The Internet Archive still has a copy here. However, you can still download ROUGE 1.5.5 from andersjo/pyrouge.

You can compute the ROUGE scores between a candidate text file and a ground-truth text file where each file contains one summary per line with the following command:

```
python -c "import helpers; helpers.test_rouge('tmp', 'save_gold.txt', 'save_pred.txt')"
```

1.5.1 Two flavors of ROUGE-L

In the ROUGE paper, two flavors of ROUGE-L are described:

1. sentence-level: Compute longest common subsequence (LCS) between two pieces of text. Newlines are ignored. This is called rougeL in this package.
2. summary-level: Newlines in the text are interpreted as sentence boundaries, and the LCS is computed between each pair of reference and candidate sentences, and something called union-LCS is computed. This is called `rougeLsum` in the rouge-score package.

Both ROUGE-L and ROUGE-L-SUM are calculated in this library.

CHAPTER

TWO

GETTING STARTED

2.1 Install

Installation is made easy due to conda environments. Simply run this command from the root project directory: `conda env create --file environment.yml` and conda will create and environment called `transformersum` with all the required packages from `environment.yml`. The spacy `en_core_web_sm` model is required for the `convert_to_extractive.py` script to detect sentence boundaries.

2.1.1 Step-by-Step Instructions

1. Clone this repository: `git clone https://github.com/HHousen/transformersum.git`.
2. Change to project directory: `cd transformersum`.
3. Run installation command: `conda env create --file environment.yml` (and activate with `conda activate transformersum`.
4. **(Optional)** If using the `convert_to_extractive.py` script then download the `en_core_web_sm` spacy model: `python -m spacy download en_core_web_sm`.

2.2 Tutorial

2.2.1 I just want to summarize some text

GUI

If all you want to do is summarize a text string using a pre-trained model then follow the below steps:

1. Download a summarization model. Link to *pre-trained extractive models*. Link to *pre-trained abstractive models*.
2. Put the model in a folder named `models` in the project root.
3. Run `python predictions_website.py` and open the link.
4. On the website enter your text, select your downloaded model, and click “SUBMIT”.

Programmatically

If you want to summarize text using a pre-trained model from python code then follow the below steps:

1. Download a summarization model. Link to *pre-trained extractive models*. Link to *pre-trained abstractive models*.
2. Instantiate the model:

   For extractive summarization:

   ```
   from extractive import ExtractiveSummarizer
   model = ExtractiveSummarizer.load_from_checkpoint("path/to/ckpt/file")
   ```

   For abstractive summarization:

   ```
   from abstractive import AbstractiveSummarizer
   model = AbstractiveSummarizer.load_from_checkpoint("path/to/ckpt/file")
   ```

3. Run prediction on a string of text:

   ```
   text_to_summarize = "Something Awesome"
   summary = model.predict(text_to_summarize)
   ```

Important: When loading a pre-trained model you may encounter this common error:

```
RuntimeError: Error(s) in loading state_dict for ExtractiveSummarizer:
Missing key(s) in state_dict: "word_embedding_model.embeddings.position_ids".
```

To solve this issue, set `strict=False` like so: `model = ExtractiveSummarizer.load_from_checkpoint("distilroberta-base-ext-sum.ckpt", strict=False)`.

Note: If you are using an *`ExtractiveSummarizer`*, then you can pass `num_summary_sentences` to specify the number of sentences in the output summary. For instance, `summary = model.predict(text_to_summarize, num_summary_sentences=5)`. The default is 3 sentences. More info at *`extractive.ExtractiveSummarizer.predict()`*.

2.2.2 Extractive Summarization

Outline:

1. Convert dataset from abstractive to extractive (different for each dataset)
2. Create features and tokenize the extractive dataset (different for every model)
3. Train the model using the *training script*
4. Test the model using the *training script*

Lets train a model that performs extractive summarization. In this tutorial we will be using BERT, but you can easily use any autoencoding model from huggingface/transformers.

Note: Autoencoding models are pretrained by corrupting the input tokens in some way and trying to reconstruct the original sentence. They correspond to the encoder of the original transformer model in the sense that they get access to the full inputs without any mask. Those models usually build a bidirectional representation of the whole sentence. They can be fine-tuned and achieve great results on many tasks such as text generation, but their most natural application is sentence classification or token classification. A typical example of such models is BERT. For more information about the different type of transformer models go to the Huggingface "Summary of the models" page.

The first step to train this model is to download the data. You can see all the datasets that are natively supported on the *Extractive Supported Datasets* page. We will be using the well-known *CNN/DailyMail dataset* since its summaries are relatively extractive and the input articles are not incredibly long. You can download the data from the "Data Download Link" link on the *CNN/DM* page. You can skip directly to step 2 as listed above by downloading the "Extractive Version" or you can skip to step 3 by downloading the `bert-base-uncased-ext-sum` model data from *CNN/DM*.

To be clear, this is an abstractive dataset so we will convert it to the extractive task using the `convert_to_extractive.py` script. You can read more about this script on the *Convert Abstractive to Extractive Dataset* page, but in short it creates a completely extractive summary that maximizes the ROUGE score between itself and the ground-truth abstractive summary. Labels (a list of 0s and 1s where 0s correspond to sentences that should not be in the summary and 1s correspond to sentences that should be in the summary) can be generated from this extractive summary. Visit *Convert Abstractive to Extractive Dataset* if you want to do this to your own dataset. For now, all you need to understand is that the above happens. You can download the preprocessed data instead of recomputing and recreating it yourself. However, there is one more step you can skip by downloading preprocessed data.

Command to convert dataset to extractive (*more info*):

```
python convert_to_extractive.py ./datasets/cnn_dailymail_processor/cnn_dm --shard_
↪interval 5000 --compression --add_target_to test
```

Once we have an extractive dataset, we need to convert the text into features that the computer can understand. This includes `input_ids`, `attention_mask`, `sent_rep_token_ids`, and more. The *`extractive.ExtractiveSummarizer.forward()`* and *`data.SentencesProcessor.get_features()`* docstrings explains these features nicely. The huggingface/transformers glossary is a good resource as well. This conversion to model-specific features happens automatically before training begins. Since the features are model-specific, the training script is responsible for converting the data. It creates a *`SentencesProcessor`* that does most of the heavy lifting. You can learn more about this automatic preprocessing on the *Automatic Preprocessing* page.

Command to only pre-process the data and stop right before training would begin (*more info*):

```
python main.py --data_path ./datasets/cnn_dailymail_processor/cnn_dm --use_logger␣
↪tensorboard --model_name_or_path bert-base-uncased --model_type bert --do_train --only_
↪preprocess
```

If you didn't run the above commands then download the `bert-base-uncased-ext-sum` model data from *CNN/DM*. You can do this from the command line with `gdown <link_to_data>` (install `gdown` with `pip install gdown`). Extract the data with `tar -xzvf bert-base-uncased.tar.gz`. Now you are ready to train. The BERT model will be downloaded automatically by the `huggingface/transformers` library.

Training command:

```
python main.py \
--model_name_or_path bert-base-uncased \
--model_type bert \
--data_path ./bert-base-uncased \
--max_epochs 3 \
--accumulate_grad_batches 2 \
--warmup_steps 2300 \
--gradient_clip_val 1.0 \
--optimizer_type adamw \
```

(continues on next page)

(continued from previous page)

```
--use_scheduler linear \
--do_train --do_test \
--batch_size 16
```

You can learn more about the above command on *Training an Extractive Summarization Model*.

Important: More example training commands can be found on the TransformerSum Weights & Biases page. Just click the name of a training run, go to the overview page by clicking the "i" icon in the top left, and look at the command value.

2.2.3 Abstractive Summarization

Lets train a model that performs abstractive summarization. Whereas autoencoding models are used for extractive summarization, sequence-to-sequence (seq2seq) models are used for abstractive summarization. In short, autoregressive models correspond to the decoder of the original transformer model, autoencoding models correspond to the encoder, and sequence-to-sequence models use both the encoder and the decoder of the original transformer.

Note: Sequence-to-sequence models use both the encoder and the decoder of the original transformer, either for translation tasks or by transforming other tasks to sequence-to-sequence problems. They can be fine-tuned to many tasks but their most natural applications are translation, summarization and question answering. The original transformer model is an example of such a model (only for translation), T5 is an example that can be fine-tuned on other tasks.

You can easily fine-tune a seq2seq model on a summarization dataset using the summarization examples in huggingface/transformers. Thus, in this project we focus on being able to use any autoencoding model with a autoregressive model to create an EncoderDecoderModel. We also focus on performing *abstractive summarization on long sequences* (or *see the below short explanation*).

In this tutorial we will be constructing bert-to-bert, but you can easily use a different model combination from huggingface/transformers. The `--model_name_or_path` option specifies the encoder and the `--decoder_model_name_or_path` specifies the decoder. If `--decoder_model_name_or_path` is not set then the value of `--model_name_or_path` is used for the decoder.

Any summarization dataset from huggingface/nlp can be used for training by only changing 4 options (specifically `--dataset`, `--dataset_version`, `--data_example_column`, and `--data_summarized_column`). The `nlp` library will handle downloading and pre-processing while the `abstractive.py` script will handle tokenization automatically. The CNN/DM dataset is the default so if you want to use that dataset you don't need to specify any options concerning data. There is a list of suggested datasets at *Abstractive Supported Datasets*.

So, in brief, training an abstractive model is as easy as running one command. Go to *Example* for an example training command.

Long Sequences Abstractive - Quick Tutorial: Set `--model_name_or_path` to `allenai/led-base-16384` or `allenai/led-large-16384`. You can now create summaries from sequences up to 16,384 tokens using the LED.

2.3 Long Sequence Summarization

This project can summarize long sequences (where long sequences are considered those greater than 512-1024 tokens) using both extractive and abstractive models.

To perform **extractive summarization** on long sequences, simply use the `longformer` model as the `word_embedding_model`, which is specified by `--model_name_or_path`. In other words, set `--model_name_or_path` to `allenai/longformer-base-4096` or `allenai/longformer-large-4096` to summarize documents of max length 4,096 tokens. For the most up-to-date model shortcut codes visit the huggingface pretrained models page and the community models page.

To perform **abstractive summarization** on long sequences, simply use the `LED` (LongformerEncoderDecoder) model, which is specified by `--model_name_or_path`. In other words, set `--model_name_or_path` to `allenai/led-base-16384` or `allenai/led-large-16384` to summarize documents of max length 16,384 tokens. For the most up-to-date model shortcut codes visit the community models page. Long abstractive summarization used to require a complicated setup with specific versions of three separate libraries. But, as of `huggingface/transformers` v4.2.0, the `LED` was incorporated directly into the library, thus simplifying the fine-tuning process.

Abstractive text summarization is a sequence-to-sequence problem solved by sequence-to-sequence models. However, state-of-the-art seq2seq models only function on short sequences. Nevertheless, BART can be modified to use the sliding window attention from the longformer to create a seq2seq model that can abstractively summarize sequences up to 16,384 tokens. Visit *Abstractive Long Summarization* for more information.

CHAPTER

THREE

MAIN SCRIPT

The same script is used to train, validate, and test both extractive and abstractive models. The `--mode` argument switches between using *ExtractiveSummarizer* and *AbstractiveSummarizer*, with *ExtractiveSummarizer* as the default.

Note: The below `--help` output only shows the generic commands that can be used for both extractive and abstractive models. Run the command with the `--mode` set to see the commands specific to each summarization technique. The `--help` output for each is also in this documentation: *Extractive* and *Abstractive*

All training arguments can be found in the pytorch_lightning trainer documentation.

Output of `python main.py --help`:

```
usage: main.py [-h] [--mode {extractive,abstractive}]
               [--default_root_dir DEFAULT_ROOT_DIR]
               [--weights_save_path WEIGHTS_SAVE_PATH] [--learning_rate LEARNING_RATE]
               [--min_epochs MIN_EPOCHS] [--max_epochs MAX_EPOCHS]
               [--min_steps MIN_STEPS] [--max_steps MAX_STEPS]
               [--accumulate_grad_batches ACCUMULATE_GRAD_BATCHES]
               [--check_val_every_n_epoch CHECK_VAL_EVERY_N_EPOCH] [--gpus GPUS]
               [--gradient_clip_val GRADIENT_CLIP_VAL] [--overfit_batches OVERFIT_
↪BATCHES]
               [--train_percent_check TRAIN_PERCENT_CHECK]
               [--val_percent_check VAL_PERCENT_CHECK]
               [--test_percent_check TEST_PERCENT_CHECK] [--amp_level AMP_LEVEL]
               [--precision PRECISION] [--seed SEED] [--profiler]
               [--progress_bar_refresh_rate PROGRESS_BAR_REFRESH_RATE]
               [--num_sanity_val_steps NUM_SANITY_VAL_STEPS]
               [--use_logger {tensorboard,wandb}] [--wandb_project WANDB_PROJECT]
               [--gradient_checkpointing] [--do_train] [--do_test]
               [--load_weights LOAD_WEIGHTS]
               [--load_from_checkpoint LOAD_FROM_CHECKPOINT]
               [--resume_from_checkpoint RESUME_FROM_CHECKPOINT]
               [--use_custom_checkpoint_callback]
               [--custom_checkpoint_every_n CUSTOM_CHECKPOINT_EVERY_N]
               [--no_wandb_logger_log_model]
               [--auto_scale_batch_size AUTO_SCALE_BATCH_SIZE] [--lr_find]
               [--adam_epsilon ADAM_EPSILON] [--optimizer_type OPTIMIZER_TYPE]
               [--ranger-k RANGER_K] [--warmup_steps WARMUP_STEPS]
               [--use_scheduler USE_SCHEDULER] [--end_learning_rate END_LEARNING_RATE]
               [--weight_decay WEIGHT_DECAY] [-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
```

(continues on next page)

(continued from previous page)

```
optional arguments:
-h, --help            show this help message and exit
--mode {extractive,abstractive}
                        Extractive or abstractive summarization training. Default is
                        'extractive'.
--default_root_dir DEFAULT_ROOT_DIR
                        Default path for logs and weights. To use this option with␣
↪the
                        `wandb` logger specify the `--no_wandb_logger_log_model`␣
↪option.
--weights_save_path WEIGHTS_SAVE_PATH
                        Where to save weights if specified. Will override
                        `--default_root_dir` for checkpoints only. Use this if for
                        whatever reason you need the checkpoints stored in a␣
↪different
                        place than the logs written in `--default_root_dir`. This␣
↪option
                        will override the save locations when using a custom␣
↪checkpoint
                        callback, such as those created when using
                        `--use_custom_checkpoint_callback or
                        `--custom_checkpoint_every_n`. If you are using the `wandb`
                        logger, then you must also set `--no_wandb_logger_log_model`␣
↪when
                        using this option. Model weights are saved with the wandb␣
↪logs to
                        be uploaded to wandb.ai by default. Setting this option␣
↪without
                        setting `--no_wandb_logger_log_model` effectively creates two
                        save paths, which will crash the script.
--learning_rate LEARNING_RATE
                        The initial learning rate for the optimizer.
--min_epochs MIN_EPOCHS
                        Limits training to a minimum number of epochs
--max_epochs MAX_EPOCHS
                        Limits training to a max number number of epochs
--min_steps MIN_STEPS
                        Limits training to a minimum number number of steps
--max_steps MAX_STEPS
                        Limits training to a max number number of steps
--accumulate_grad_batches ACCUMULATE_GRAD_BATCHES
                        Accumulates grads every k batches. A single step is one␣
↪gradient
                        accumulation cycle, so setting this value to 2 will cause 2
                        batches to be processed for each step.
--check_val_every_n_epoch CHECK_VAL_EVERY_N_EPOCH
                        Check val every n train epochs.
--gpus GPUS           Number of GPUs to train on or Which GPUs to train on. (default:
                        -1 (all gpus))
--gradient_clip_val GRADIENT_CLIP_VAL
                        Gradient clipping value
```

(continues on next page)

(continued from previous page)

```
  --overfit_batches OVERFIT_BATCHES
                        Uses this much data of all datasets (training, validation,␣
↪test).
                        Useful for quickly debugging or trying to overfit on purpose.
  --train_percent_check TRAIN_PERCENT_CHECK
                        How much of training dataset to check. Useful when debugging␣
↪or
                        testing something that happens at the end of an epoch.
  --val_percent_check VAL_PERCENT_CHECK
                        How much of validation dataset to check. Useful when␣
↪debugging or
                        testing something that happens at the end of an epoch.
  --test_percent_check TEST_PERCENT_CHECK
                        How much of test dataset to check.
  --amp_level AMP_LEVEL
                        The optimization level to use (O1, O2, etc...) for 16-bit GPU
                        precision (using NVIDIA apex under the hood).
  --precision PRECISION
                        Full precision (32), half precision (16). Can be used on CPU,
↪ GPU
                        or TPUs.
  --seed SEED           Seed for reproducible results. Can negatively impact performace
                        in some cases.
  --profiler            To profile individual steps during training and assist in
                        identifying bottlenecks.
  --progress_bar_refresh_rate PROGRESS_BAR_REFRESH_RATE
                        How often to refresh progress bar (in steps). In notebooks,
                        faster refresh rates (lower number) is known to crash them
                        because of their screen refresh rates, so raise it to 50 or␣
↪more.
  --num_sanity_val_steps NUM_SANITY_VAL_STEPS
                        Sanity check runs n batches of val before starting the␣
↪training
                        routine. This catches any bugs in your validation without␣
↪having
                        to wait for the first validation check.
  --use_logger {tensorboard,wandb}
                        Which program to use for logging. If `wandb` is chosen then␣
↪model
                        weights will automatically be uploaded to wandb.ai.
  --wandb_project WANDB_PROJECT
                        The wandb project to save training runs to if `--use_logger`␣
↪is
                        set to `wandb`.
  --gradient_checkpointing
                        Enable gradient checkpointing (save memory at the expense of␣
↪a
                        slower backward pass) for the word embedding model. More␣
↪info: ht
                        tps://github.com/huggingface/transformers/pull/4659#issue-
↪4248418
                        71
```

(continues on next page)

(continued from previous page)

```
    --do_train            Run the training procedure.
    --do_test             Run the testing procedure.
    --load_weights LOAD_WEIGHTS
                            Loads the model weights from a given checkpoint
    --load_from_checkpoint LOAD_FROM_CHECKPOINT
                            Loads the model weights and hyperparameters from a given
                            checkpoint.
    --resume_from_checkpoint RESUME_FROM_CHECKPOINT
                            To resume training from a specific checkpoint pass in the␣
↪path
                            here. Automatically restores model, epoch, step, LR␣
↪schedulers,
                            apex, etc...
    --use_custom_checkpoint_callback
                            Use the custom checkpointing callback specified in `main()`␣
↪by
                            `args.checkpoint_callback`. By default this custom callback␣
↪saves
                            the model every epoch and never deletes the saved weights␣
↪files.
                            You can change the save path by setting the `--weights_save_
↪path`
                            option.
    --custom_checkpoint_every_n CUSTOM_CHECKPOINT_EVERY_N
                            The number of steps between additional checkpoints. By␣
↪default
                            checkpoints are saved every epoch. Setting this value will␣
↪save
                            them every epoch and every N steps. This does not use the␣
↪same
                            callback as `--use_custom_checkpoint_callback` but instead␣
↪uses a
                            different class called `StepCheckpointCallback`. You can␣
↪change
                            the save path by setting the `--weights_save_path` option.
    --no_wandb_logger_log_model
                            Only applies when using the `wandb` logger. Set this␣
↪argument to
                            NOT save checkpoints in wandb directory to upload to W&B␣
↪servers.
    --auto_scale_batch_size AUTO_SCALE_BATCH_SIZE
                            Auto scaling of batch size may be enabled to find the largest
                            batch size that fits into memory. Larger batch size often␣
↪yields
                            better estimates of gradients, but may also result in longer
                            training time. Currently, this feature supports two modes
↪'power'
                            scaling and 'binsearch' scaling. In 'power' scaling, starting
                            from a batch size of 1 keeps doubling the batch size until an
                            out-of-memory (OOM) error is encountered. Setting the␣
↪argument to
                            'binsearch' continues to finetune the batch size by␣
↪performing a
```

(continues on next page)

(continued from previous page)

```
                           binary search. 'binsearch' is the recommended option.
  --lr_find              Runs a learning rate finder algorithm (see
                           https://arxiv.org/abs/1506.01186) before any training, to␣
↪find
                           optimal initial learning rate.
  --adam_epsilon ADAM_EPSILON
                           Epsilon for Adam optimizer.
  --optimizer_type OPTIMIZER_TYPE
                           Which optimizer to use: `adamw` (default), `ranger`,␣
↪`qhadam`,
                           `radam`, or `adabound`.
  --ranger-k RANGER_K   Ranger (LookAhead) optimizer k value (default: 6). LookAhead
                           keeps a single extra copy of the weights, then lets the
                           internalized 'faster' optimizer (for Ranger, that's RAdam)
                           explore for 5 or 6 batches. The batch interval is specified␣
↪via
                           the k parameter.
  --warmup_steps WARMUP_STEPS
                           Linear warmup over warmup_steps. Only active if `--use_
↪scheduler`
                           is set to linear.
  --use_scheduler USE_SCHEDULER
                           Three options: 1. `linear`: Use a linear schedule that␣
↪inceases
                           linearly over `--warmup_steps` to `--learning_rate` then
                           decreases linearly for the rest of the training process. 2.
                           `onecycle`: Use the one cycle policy with a maximum learning␣
↪rate
                           of `--learning_rate`. (default: False, don't use any␣
↪scheduler)
                           3. `poly`: polynomial learning rate decay from `--learning_
↪rate`
                           to `--end_learning_rate`
  --end_learning_rate END_LEARNING_RATE
                           The ending learning rate when `--use_scheduler` is poly.
  --weight_decay WEIGHT_DECAY
  -l {DEBUG,INFO,WARNING,ERROR,CRITICAL}, --log {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                           Set the logging level (default: 'Info').
```

CHAPTER

FOUR

DIFFERENCES FROM PRESUMM/BERTSUM

This project accomplishes a task similar to BertSum/PreSumm (and is based on their research). However, `TransformerSum` improves various aspects and adds several features on top of `BertSum/PreSumm`. The most notable improvements are listed below.

Note: PreSumm (Yang Liu and Mirella Lapata) builds on top of BertSum (Yang Liu) by adding abstractive summarization. BertSum was extractive only.

4.1 General

- This project uses `pytorch_lighting`, which is a template to better organize PyTorch code. It automates most of the training loop. It also results in easier to read code than plain PyTorch because everything is organized.
- `TransformerSum` contains comments explaining design decisions and how code works. Significant functions also have detailed docstrings. See *`get_features()`* for an example.
- Easy pre-trained model loading and modification thanks to `pytorch_lightning`'s `load_from_checkpoint()` that is automatically inherited by every `pl.LightningModule`.
- There is a `prediction()` function included with the *`extractive.ExtractiveSummarizer`* and *`abstractive.AbstractiveSummarizer`* classes that summaries a string. This makes it easy to initialize a model and quickly perform inference.

4.2 Converting a Dataset to Extractive

- Separation between the dataset conversion from abstractive to extractive and the training process. This means the same converted dataset can be used by any transformer model with further automatic processing.
- The `convert_to_extractive.py` script uses the more up-to-date `spacy` whereas BertSum utilizes `Stanford CoreNLP`. Spacy has the added benefit of being a python library (while CoreNLP is a java application), which means easier to understand code.
- Supports the same *`greedy_selection()`* and *`combination_selection()`* of `BertSum`.
- A more robust CLI that allows for various desired outputs.
- Built in optional gzip compression.
- Integration with huggingface/nlp means any summarization dataset in the `nlp` library can be converted to extractive by only modifying 4 options (specifically `--dataset`, `--dataset_version`, `--data_example_column`, and `--data_summarized_column`). The `nlp` library will handle downloading and pre-processing.

4.3 Pooling for Extractive Models

- Supports the `sent_rep_tokens` and `mean_tokens` pooling methods whereas `BertSum` only supports `sent_rep_tokens`. See *Pooling Modes* for more info.
- Pooling is separated from the main model's `forward()` function for easy extendability.

4.4 Data for Extractive Models

- The data processing and loading uses normal PyTorch `DataLoader``s and ``Dataset``s instead of recreations of these classes as in ``BertSum`.
- A `collate_fn` function converts the data from the dataset to tensors suitable for input to the model, as is stand PyTorch convention.
- The `collate_fn` function also performs "smart batching." It performs padding on the necessary information for each batch, which is more efficient than padding for the entire dataset or each chunk.
- The *`data.FSIterableDataset`* class loads a dataset in chunks and is a subclass of `torch.utils.data.IterableDataset`. While `BertSum` also supports chunked loading to lower RAM usage, `TransformerSum`'s technique is more robust and directly integrates with PyTorch.

4.5 Extractive Model and Training

- **Compatible with every huggingface/transformers transformer encoder model.** `BertSum` can only use Bert, whereas this project supports all encoders by only changing two options when training.
- Easily extendable with new custom models that are saved in the `huggingface/transformers` format. In this way, integration with the `longformer` was easily accomplished (this integration was since removed since `huggingface/transformers` implemented the `longformer` directly..
- For the extractive component, `BertSum` supports three classifiers: a linear layer, a transformer, and a LSTM network. This project supports four different classifiers: `linear`, a few linear layers with dropout and an activation function; `simple_linear`, a single linear layer; `transformer`, which runs the input through some `nn.TransformerEncoderLayer` layers; and `transformer_linear`, a linear layer combined with a transformer. The classifier is responsible for removing the hidden features from each sentence embedding and converting them to a single number. The BertSum paper indicates that the difference between these classifiers is not major.
- The reduction method for the BCE loss function is different in `TransformerSum` than *BertSum*. *BertSum* takes the sum of the losses for each sentence (ignoring padding) even though it looks like it uses the mean. Five different reduction methods were tested (see the *Loss Functions*). There did not appear to a significant difference, but the best was chosen.
- The batch size parameter of `BertSum` is not the real batch size (which is likely caused by the custom `DataLoader`). In this project batch size is the number of documents processed on the GPU at once.
- Multiple optimizers are supported "out-of-the-box" in `TransformerSum` without any need to modify the code.
- The `OneCycle` and `linear_schedule_with_warmup` schedulers are supported in `TransformerSum` "out-of-the-box."
- Logging of all five loss functions (for both the train and validation sets), accuracy, and more is supported. Weights & Biases and Tensorboard are supported "out-of-the-box" but `pytorch_lightning` can integrate several other loggers.

4.6 Abstractive Model and Training

- Dataset preparation happens extremely quickly (minutes instead of hours; CNN/DM can be ready to train in about 10 minutes from the raw data)
- Integration with huggingface/nlp means any summarization dataset in the `nlp` library can be used for training by only modifying 4 options (specifically `--dataset`, `--dataset_version`, `--data_example_column`, and `--data_summarized_column`). The `nlp` library will handle downloading and pre-processing while the `abstractive.py` script will handle tokenization automatically.
- **Compatible with every huggingface/transformers EncoderDecoder model.** `PreSumm` only supports a BERT encoder and a standard transformer decoder, whereas this project supports all EncoderDecoder models by changing a single option (`--model_name_or_path`).

4.7 Where `BertSum` is Better

- `BertSum` has an LSTM classifier, which `TransformerSum` does not replicate.

CHAPTER

FIVE

GENERAL API REFERENCE

This page contains the API reference for modules that contain code used for both the extractive and abstractive summarization components.

5.1 Helpers

`class helpers.LabelSmoothingLoss`(*label_smoothing*, *tgt_vocab_size*, *ignore_index=- 100*)

CrossEntropyLoss with label smoothing, KL-divergence between q_{smoothed ground truth prob.}(w) and p_{prob. computed by model}(w) is minimized. From OpenNMT with modifications: https://github.com/OpenNMT/OpenNMT-py/blob/e8622eb5c6117269bb3accd8eb6f66282b5e67d9/onmt/utils/loss.py#L186

forward(*output*, *target*)

output (FloatTensor): batch_size x n_classes target (LongTensor): batch_size

training: bool

`class helpers.SortishSampler`(*data*, *batch_size*, *pad_token_id*)

Go through the text data by order of src length with a bit of randomness. From fastai repo with modifications.

key(*i*)

`class helpers.StepCheckpointCallback`(*step_interval=1000*, *save_name='model'*, *save_path='.'*, *num_saves_to_keep=5*)

on_batch_end(*trainer*, *pl_module*)

Called when the training batch ends.

`helpers.block_trigrams`(*candidate*, *prediction*)

Decrease repetition in summaries by checking if a trigram from `prediction` exists in `candidate`

Parameters

- **candidate** (*str*) – The string to check for trigrams from `prediction`
- **prediction** (*list*) – A list of strings to extract trigrams from

Returns True if overlapping trigrams detected, False otherwise.

Return type bool

`helpers.generic_configure_optimizers`(*hparams*, *train_dataloader*, *params_to_update*)

Configure the optimizers. Returns the optimizer and scheduler specified by the values in `hparams`. This is a generic function that both the extractive and abstractive scripts use.

`helpers.get_optimizer`(*hparams*, *optimizer_grouped_parameters*) `helpers.`**`load_json`**(*json_file*)

Load a json file even if it is compressed with gzip.

Parameters **`json_file`** (*`str`*) – Path to json file

Returns (documents, file_path), loaded json and path to file

Return type tuple

`helpers.`**`lr_lambda_func`**(*current_step*, *num_warmup_steps*, *num_training_steps*)

`helpers.`**`pad`**(*data*, *pad_id*, *width=None*, *pad_on_left=False*, *nearest_multiple_of=False*)

Pad `data` with `pad_id` to `width` on the right by default but if `pad_on_left` then left.

`helpers.`**`pad_tensors`**(*tensors*, *pad_id=0*, *width=None*, *pad_on_left=False*, *nearest_multiple_of=False*)

Pad `tensors` with `pad_id` to `width` on the right by default but if `pad_on_left` then left.

`helpers.`**`test_rouge`**(*temp_dir*, *cand*, *ref*)

Compute ROUGE scores using the official ROUGE 1.5.5 package. This function uses the `pyrouge` python module to interface with the office ROUGE script. There should be a "<q>" token between each sentence in the `cand` and `ref` files. `pyrouge` splits sentences based on newlines but we cannot store all the summaries easily in a single text file if there is a newline between each sentence since newlines mark new summaries. Thus, the "<q>" token is used in the text files and is converted to a newline in this function. Using "<q>" instead of `\\n` also makes it easier to store the ground-truth summaries in the `convert_to_extractive.py` script.

Parameters

- **`temp_dir`** (*`str`*) – A temporary folder to store files for input to the ROUGE script.
- **`cand`** (*`str`*) – The path to the file containing one candidate summary per line with "<q>" tokens in between each sentence.
- **`ref`** (*`str`*) – The path to the file containing one ground-truth/gold summary per line with "<q>" tokens in between each sentence.

Returns Results from the ROUGE script as a python dictionary.

Return type dict

CHAPTER

SIX

EXTRACTIVE PRE-TRAINED MODELS & RESULTS

The recommended model to use is `distilroberta-base-ext-sum` because of its fast performance, relatively low number of parameters, and good performance.

6.1 Notes

The distil* models are of special significance. Distil* is a class of compressed models that started with DistilBERT. DistilBERT stands for Distillated-BERT. DistilBERT is a small, fast, cheap and light Transformer model based on Bert architecture. It has 40% less parameters than `bert-base-uncased`, runs 60% faster while preserving 99% of BERT's performances as measured on the GLUE language understanding benchmark. DistilBERT is a smaller Transformer model that bears a lot of similarities with the original BERT model while being lighter, smaller and faster to run. DistilRoBERTa reaches 95% of RoBERTa-base's performance on GLUE and is twice as fast as RoBERTa while being 35% smaller. More info at huggingface/transformers.

The remarkable performance to size ratio of the distil* models can be transferred to summarization. `distilroberta` is recommended over `distilbert` because of the architecture improvements that the original RoBERTa brought over the original BERT. Essentially, `distilroberta` is more modern than `distilbert`.

MobileBERT is similar to `distilbert` in that it is a smaller version of BERT that achieves amazing performance at a very small size. According to the authors, MobileBERT is *2.64x smaller and 2.45x faster* than DistilBERT. DistilBERT successfully halves the depth of BERT model by knowledge distillation in the pre-training stage and an optional fine-tuning stage. MobileBERT only uses knowledge transfer in the pre-training stage and does not require a fine-tuned teacher or data augmentation in the down-stream tasks. DistilBERT compresses BERT by reducing its depth, while MobileBERT compresses BERT by reducing its width, which has been shown to be more effective. MobileBERT usually needs a larger learning rate and more training epochs in fine-tuning than the original BERT.

Important: Interactive charts, graphs, raw data, run commands, hyperparameter choices, and more for all trained models are publicly available on the TransformerSum Weights & Biases page. You can download the raw data for each model on this site, or download an overview as a CSV. Please open an issue if you have questions about these models.

Additionally, all of the models on this page were trained completely for free using Tesla P100-PCIE-16GB GPUs on Google Colaboratory. Those that took over 12 hours to train were split into multiple training sessions since `pytorch_lightning` enables easy resuming with the `--resume_from_checkpoint` argument.

6.2 CNN/DM

Name	Comments	Model Download	Data Download
distilbert-base-uncased-ext-sum	None	Model & All Checkpoints	CNN/DM Bert Uncased
distilroberta-base-ext-sum	None	Model & All Checkpoints	CNN/DM Roberta
bert-base-uncased-ext-sum	None	Model & All Checkpoints	CNN/DM Bert Uncased
roberta-base-ext-sum	None	Model & All Checkpoints	CNN/DM Roberta
bert-large-uncased-ext-sum	None	Not yet…	CNN/DM Bert Uncased
roberta-large-ext-sum	None	Not yet…	CNN/DM Roberta
longformer-base-4096-ext-sum	None	Not yet…	CNN/DM Longformer
mobilebert-uncased-ext-sum	Trained with lr=8e-5	Model & All Checkpoints	CNN/DM Bert Uncased

6.2.1 CNN/DM ROUGE Scores

Test set results on the CNN/DailyMail dataset using ROUGE F_1.

Name	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-L-Sum
distilbert-base-uncased-ext-sum	42.71	19.91	27.52	39.18
distilroberta-base-ext-sum	42.87	20.02	27.46	39.31
bert-base-uncased-ext-sum	42.78	19.83	27.43	39.18
roberta-base-ext-sum	43.24	20.36	27.64	39.65
bert-large-uncased-ext-sum	Not yet…	Not yet…	Not yet…	Not yet…
roberta-large-ext-sum	Not yet…	Not yet…	Not yet…	Not yet…
longformer-base-4096-ext-sum	Not yet…	Not yet…	Not yet…	Not yet…
mobilebert-uncased-ext-sum	42.01	19.31	26.89	38.53

Note: Currently, `distilbert` beats `bert-base-uncased` by 1.0014% (`(42.71/42.78+19.91/19.83+27.52/27.43+39.18/39.18)/4=1.0014197729882865`). Since `bert-base-uncased` has more parameters than `distilbert`, this is unusual and is likely a tuning issue. This suggests that tuning the hyperparameters of `bert-base-uncased` can improve its performance. `distilroberta` matches 92.7% of the performance of `roberta-base` (`(42.87/43.24+20.02/20.36+27.46/27.64+29.31/39.65)/4=0.9268623888753363`).

Important: `mobilebert-uncased-ext-sum` achieves 96.59% (`(42.01/43.25+19.31/20.24+38.53/39.63)/3`) of the performance of BertSum while containing 4.45 times (`109483009/24582401`) fewer parameters. It achieves 94.06% (`(42.01/44.41+19.31/20.86+38.53/40.55)/3`) of the performance of MatchSum (Zhong et al., 2020), the current extractive state-of-the-art.

6.2.2 CNN/DM Training Times and Model Sizes

Name	Time	Model Size
distilbert-base-uncased-ext-sum	6h 22m 32s	796.4MB
distilroberta-base-ext-sum	6h 21m 37s	980.8MB
bert-base-uncased-ext-sum	12h 51m 17s	1.3GB
roberta-base-ext-sum	13h 7m 3s	1.5GB
bert-large-uncased-ext-sum	Not yet…	Not yet…
roberta-large-ext-sum	Not yet…	Not yet…
longformer-base-4096-ext-sum	Not yet…	Not yet…
mobilebert-uncased-ext-sum	8h 26m 32s	295.6MB

Important: `distilroberta-base-ext-sum` trains in about 6.5 hours on 1 P100-PCIE-16GB GPU, while MatchSum, the current state-of-the-art in extractive summarization on CNN/DM, takes 30 hours on 8 Tesla-V100-16G GPUs to train. If a V100 is about 2x as powerful as a P100, then it would take 480 hours (`30*8*2`) to train MatchSum on one P100. This simplistic approximation suggests that it takes about 74x (`480/6.5`) more time to train MatchSum than `distilroberta-base-ext-sum`.

6.3 WikiHow

Name	Comments	Model Download	Data Download
distilbert-base-uncased-ext-sum	None	Model & All Checkpoints	WikiHow Bert Uncased
distilroberta-base-ext-sum	None	Model & All Checkpoints	WikiHow Roberta
bert-base-uncased-ext-sum	None	Model & All Checkpoints	WikiHow Bert Uncased
roberta-base-ext-sum	None	Model & All Checkpoints	WikiHow Roberta
bert-large-uncased-ext-sum	None	Not yet…	WikiHow Bert Uncased
roberta-large-ext-sum	None	Not yet…	WikiHow Roberta
mobilebert-uncased-ext-sum	Trained with lr=8e-5	Model & All Checkpoints	WikiHow Bert Uncased

6.3.1 WikiHow ROUGE Scores

Test set results on the WikiHow dataset using ROUGE F_1.

Name	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-L-Sum
distilbert-base-uncased-ext-sum	30.69	8.65	19.13	28.58
distilroberta-base-ext-sum	31.07	8.96	19.34	28.95
bert-base-uncased-ext-sum	30.68	08.67	19.16	28.59
roberta-base-ext-sum	31.26	09.09	19.47	29.14
bert-large-uncased-ext-sum	Not yet…	Not yet…	Not yet…	Not yet…
roberta-large-ext-sum	Not yet…	Not yet…	Not yet…	Not yet…
mobilebert-uncased-ext-sum	30.72	8.78	19.18	28.59

Note: These are the results of an extractive model, which means they are fairly good because they come close to abstractive models. The R1/R2/RL-Sum results of a base transformer model from the PEGASUS paper are 32.48/10.53/23.86. The net difference from `distilroberta-base-ext-sum` is +1.41/+1.57/-5.09. Compared to the **abstractive** SOTA prior to PEGASUS, which was 28.53/9.23/26.54, `distilroberta-base-ext-sum` performs +2.54/-0.27/+2.41. However, the base PEGASUS model obtains scores of 36.58/15.64/30.01, which are much better than `distilroberta-base-ext-sum`, as one would expect.

6.3.2 WikiHow Training Times and Model Sizes

Name	Time	Model Size
distilbert-base-uncased-ext-sum	3h 42m 12s	796.4MB
distilroberta-base-ext-sum	4h 27m 23s	980.8MB
bert-base-uncased-ext-sum	7h 29m 06s	1.3GB
roberta-base-ext-sum	7h 35m 59s	1.5GB
bert-large-uncased-ext-sum	Not yet…	Not yet…
roberta-large-ext-sum	Not yet…	Not yet…
mobilebert-uncased-ext-sum	4h 22m 19s	295.6MB

6.4 arXiv-PubMed

Name	Comments	Model Download	Data Download
distilbert-base-uncased-ext-sum	None	Model & All Checkpoints	arXiv-PubMed Bert Uncased
distilroberta-base-ext-sum	None	Model & All Checkpoints	arXiv-PubMed Roberta
bert-base-uncased-ext-sum	None	Model & All Checkpoints	arXiv-PubMed Bert Uncased
roberta-base-ext-sum	None	Model & All Checkpoints	arXiv-PubMed Roberta
bert-large-uncased-ext-sum	None	Not yet…	arXiv-PubMed Bert Uncased
roberta-large-ext-sum	None	Not yet…	arXiv-PubMed Roberta
longformer-base-4096-ext-sum	None	Not yet…	arXiv-PubMed Longformer
mobilebert-uncased-ext-sum	None	Model & All Checkpoints	arXiv-PubMed Bert Uncased

6.4.1 arXiv-PubMed ROUGE Scores

Test set results on the arXiv-PubMed dataset using ROUGE F_1.

Name	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-L-Sum
distilbert-base-uncased-ext-sum	34.93	12.21	19.62	31.00
distilroberta-base-ext-sum	34.70	12.16	19.52	30.82
bert-base-uncased-ext-sum	34.80	12.26	19.67	30.92
roberta-base-ext-sum	34.81	12.26	19.65	30.91
bert-large-uncased-ext-sum	Not yet…	Not yet…	Not yet…	Not yet…
roberta-large-ext-sum	Not yet…	Not yet…	Not yet…	Not yet…
longformer-base-4096-ext-sum	Not yet…	Not yet…	Not yet…	Not yet…
mobilebert-uncased-ext-sum	33.97	11.74	19.63	30.19

Note: These are the results of an extractive model, which means they are fairly good because they come close to abstractive models. The R1/R2/RL-Sum results of a base transformer model from the PEGASUS paper are 34.79/7.69/19.51 (average of 35.63/7.95/20.00 (arXiv) and 33.94/7.43/19.02 (PubMed)). The net difference from `distilroberta-base-ext-sum` is +0.09/-4.47/-11.31. Compared to the **abstractive** SOTA prior to

PEGASUS, which was 41.09/14.93/23.57 (average of 41.59/14.26/23.55 (arXiv) and 40.59/15.59/23.59 (PubMed)), `distilroberta-base-ext-sum` performs -6.39/-2.77/+7.25. However, the base PEGASUS model obtains scores of 37.39/12.66/23.87 (average of 34.81/10.16/22.50 (arXiv) and 39.98/15.15/25.23 (PubMed)). The large model obtains scores of 45.10/18.59/26.75 (average of 44.70/17.27/25.80 (arXiv) and 45.49/19.90/27.69 (PubMed)) which are much better than `distilroberta-base-ext-sum`, as one would expect.

6.4.2 arXiv-PubMed Training Times and Model Sizes

Name	Time	Model Size
distilbert-base-uncased-ext-sum	06h 46m 0s	796.4MB
distilroberta-base-ext-sum	06h 33m 58s	980.8MB
bert-base-uncased-ext-sum	14h 40m 10s	1.3GB
roberta-base-ext-sum	14h 39m 43s	1.5GB
bert-large-uncased-ext-sum	Not yet…	Not yet…
roberta-large-ext-sum	Not yet…	Not yet…
longformer-base-4096-ext-sum	Not yet…	Not yet…
mobilebert-uncased-ext-sum	09h 5m 45s	295.6MB

CHAPTER SEVEN

EXTRACTIVE SUPPORTED DATASETS

Note: In addition to the below datasets, all of the *abstractive datasets* can be converted for extractive summarization and thus be used to train models. See *Option 2: Automatic pre-processing through nlp* for more information.

There are several ways to obtain and process the datasets below:

1. Download the converted extractive version for use with the training script (which will preprocess the data automatically (tokenization, etc.)). Note that all the provided extractive versions are split every 500 documents and are compressed. You will have to manually process if you desire different chunk sizes.
2. Download the processed abstractive version. This is the original data after being run through its respective processor located in the `datasets` folder.
3. Download the original data in its original form, which depends on how it was obtained in the original paper.

The table under each heading contains quick links to download the data. Beneath that are instructions to process the data manually.

7.1 CNN/DM

The **CNN/DailyMail** (Hermann et al., 2015) dataset contains 93k articles from the CNN, and 220k articles the Daily Mail newspapers. Both publishers supplement their articles with bullet point summaries. Non-anonymized variant in See et al. (2017).

Type	Link
Processor Repository	artmatsak/cnn-dailymail
Data Download Link	CNN/DM official website
Processed Abstractive Dataset	Google Drive
Extractive Version	Google Drive

Download and unzip the stories directories from here for both CNN and Daily Mail. The files can be downloaded from the terminal with *gdown*, which can be installed with *pip install gdown*.

```
pip install gdown
gdown https://drive.google.com/uc?id=0BwmD_VLjROrfTHk4NFg2SndKcjQ
gdown https://drive.google.com/uc?id=0BwmD_VLjROrfM1BxdkxVaTY2bWs
tar zxf cnn_stories.tgz
tar zxf dailymail_stories.tgz
```

Note: The above Google Drive links may be outdated depending on the time you are reading this. Check the CNN/DM official website for the most up-to-date download links.

Next, run the processing code in the git submodule for artmatsak/cnn-dailymail located in `datasets/cnn_dailymail_processor`. Run `python make_datafiles.py /path/to/cnn/stories /path/to/dailymail/stories`, replacing */`path/to/cnn/stories`* with the path to where you saved the `cnn/stories` directory that you downloaded; similarly for `dailymail/stories`.

For each of the URL lists (`all_train.txt`, `all_val.txt` and `all_test.txt`) in `cnn_dailymail_processor/url_lists`, the corresponding stories are read from file and written to text files `train.source`, `train.target`, `val.source`, `val.target`, and `test.source` and `test.target`. These will be placed in the newly created `cnn_dm` directory.

The original processing code is available at abisee/cnn-dailymail, but for this project the artmatsak/cnn-dailymail processing code is used since it does not tokenize and writes the data to text file `train.source`, `train.target`, `val.source`, `val.target`, `test.source` and `test.target`, which is the format expected by `convert_to_extractive.py`.

7.2 WikiHow

WikiHow (Koupaee and Wang, 2018) is a large-scale dataset of instructions from the online WikiHow.com website. Each of 200k examples consists of multiple instruction-step paragraphs along with a summarizing sentence. The task is to generate the concatenated summary-sentences from the paragraphs.

Dataset Size	230,843
Average Article Length	579.8
Average Summary Length	62.1
Vocabulary Size	556,461

Type	Link
Processor Repository	HHousen/WikiHow-Dataset (Original Repo)
Data Download Link	wikihowAll.csv (mirror) and wikihowSep.csv
Processed Abstractive Dataset	Google Drive
Extractive Version	Google Drive

Processing Steps:

1. Download wikihowAll.csv (main repo for most up-to-date links) to `datasets/wikihow_processor`
2. Run `python process.py` (runtime: 2m), which will create a new directory called `wikihow` containing the `train.source`, `train.target`, `val.source`, `val.target`, `test.source` and `test.target` files necessary for *convert_to_extractive.py*.

7.3 PubMed/ArXiv

ArXiv and PubMed (Cohan et al., 2018) are two long document datasets of scientific publications from [arXiv.org](http://arxiv.org/) (113k) and PubMed (215k). The task is to generate the abstract from the paper body.

Datasets	# docs	avg. doc. length (words)	avg. summary length (words)
CNN	92K	656	43
Daily Mail	219K	693	52
NY Times	655K	530	38
PubMed (this dataset)	133K	3016	203
arXiv (this dataset)	215K	4938	220

Type	Link
Processor Repository	HHousen/ArXiv-PubMed-Sum (Original Repo)
Data Download Link	PubMed (mirror) and ArXiv (mirror)
Processed Abstractive Dataset	Google Drive
Extractive Version	Google Drive

Processing Steps:

1. Download PubMed and ArXiv (main repo for most up-to-date links) to `datasets/arxiv-pubmed_processor`
2. Run the command `python process.py <arxiv_articles_dir> <pubmed_articles_dir>` (runtime: 5-10m), which will create a new directory called `arxiv-pubmed` containing the `train.source`, `train.target`, `val.source`, `val.target`, `test.source` and `test.target` files necessary for *convert_to_extractive.py*.

See the repository's README.md.

Note: To convert this dataset to extractive it is recommended to use the `--sentencizer` option due to the size of the dataset. Additionally, the `--max_sentence_ntokens` should be set to `300` and the `--max_example_nsents` should be set to `600`. See the *Convert Abstractive to Extractive Dataset* section for more information. The full command should be similar to:

```
python convert_to_extractive.py ./datasets/arxiv-pubmed_processor/arxiv-pubmed \
--shard_interval 5000 \
--sentencizer \
--max_sentence_ntokens 300 \
--max_example_nsents 600
```

CHAPTER

EIGHT

CONVERT ABSTRACTIVE TO EXTRACTIVE DATASET

8.1 Overview

This script will reformat an abstractive summarization dataset to be used for extractive summarization by determining the best extractive summary that maximizes ROUGE scores. It can be used on a dataset composed of the following file structure: `train.source`, `train.target`, `val.source`, `val.target`, `test.source`, and `test.target` where each file contains one example per line and the lines of every `.train` file correspond to the lines in the respective `.target`. All the datasets on the *Extractive Supported Datasets* page will be processed to this format. You can also process any dataset contained in the huggingface/nlp library. If you use a dataset this way, the downloading and pre-processing will happen automatically.

8.2 Option 1: Manual Data Download

Simply run `convert_to_extractive.py` with the path to the data. For example, with the *CNN/DM dataset*: `python convert_to_extractive.py ./datasets/cnn_dailymail_processor/cnn_dm`. However, the recommended command is:

```
python convert_to_extractive.py ./datasets/cnn_dailymail_processor/cnn_dm --shard_
↪interval 5000 --compression --add_target_to test
```

- `--shard_interval` processes the file in chunks of `5000` and writes results to disk in chunks of `5000` (saves RAM)
- `--compression` compresses each output chunk with gzip (depending on the dataset reduces space usage requirement by about 1/2 to 1/3)
- `--add_target_to` will save the abstractive target text to the splits (in `--split_names`) specified.

The default output directory is the input directory that was specified, but the output directory can be changed with `--base_output_path` if desired.

If your files are not `train`, `val`, and `test`, then the `--split_names` argument will let you specify the correct naming pattern. The `--source_ext` and `--target_ext` let you specify the file extension of the source and target files respectively. These must be different so the process can tell each section apart.

8.3 Option 2: Automatic pre-processing through `nlp`

You will need to run the `convert_to_extractive.py` command with the `--dataset`, `--dataset_version`, `--data_example_column`, and `--data_summarized_column` options set. To use the CNN/DM dataset you would set these arguments as shown below:

```
--dataset cnn_dailymail \
--dataset_version 3.0.0 \
--data_example_column article \
--data_summarized_column highlights
```

View the help page (`python convert_to_extractive.py --help`) for more info about these options. The options are nearly identical to the *abstractive script*.

Important: All of the *abstractive datasets* can be converted for extractive summarization using this method.

The live nlp viewer visualizes the data and describes each dataset that can be used with through parameters.

8.4 Convert To Extractive Tips

Large Dataset? Need to Resume?: The `--resume` option will read the output directory and determine on which document the script left off based on the shard_file names. If `--shard_interval` was `None` then resuming is not possible. Resuming is guaranteed to produce the same output as if `--resume` was not used because of *`convert_to_extractive.check_resume_success()`*, which checks to make sure the last line in the shard file is the same as the line directly before the line to resume with.

Speed: Running Slowly? There is a `--sentencizer` option to detect sentence boundaries without parsing dependencies. Instead of loading a statistical model using `spacy`, this option will initialize the `English` Language object and add a `sentencizer` to the pipeline. This is much faster than a DependencyParser but is also less accurate since the `sentencizer` uses a simpler, rule-based strategy.

8.5 Custom Datasets

Any dataset in the format described in the *Overview* can be used with this script. Once converted, training should be the same as if using CNN/DM from *Option 1* because the *Convert To Extractive* script outputs a consistent format.

8.5.1 Extractive Dataset Format

This section briefly discusses the format of datasets created by the `convert_to_extractive` script.

The training and validation sets only need the `src` and `labels` keys saved as json. The `src` value should be a list of lists where each list contains a series of tokens (see below). The `labels` value is a list of 0s (not in summary) and 1s (sentence should be in summary) that is the same length as the `src` value (the number of sentences). Each value in this list corresponds to a sentence in `src`. The testing set is special because it needs the `src`, `labels`, and `tgt` keys. The `tgt` key represents the target summary as a single string with a <q> between each sentence.

First document in **CNN/DM** extractive **training** set:

```
{'src': [['Editor', "'s", 'note', ':', 'In', 'our', 'Behind', 'the', 'Scenes', 'series',
↪',', 'CNN', 'correspondents', 'share', 'their', 'experiences', 'in', 'covering', 'news
↪', 'and', 'analyze', 'the', 'stories', 'behind', 'the', 'events', '.'], ['Here', ',',
↪'Soledad', "O'Brien", 'takes', 'users', 'inside', 'a', 'jail', 'where', 'many', 'of',
↪'the', 'inmates', 'are', 'mentally', 'ill', '.'], ['An', 'inmate', 'housed', 'on', 'the
↪', '"', 'forgotten', 'floor', ',', '"', 'where', 'many', 'mentally', 'ill', 'inmates',
↪'are', 'housed', 'in', 'Miami', 'before', 'trial', '.'], ['MIAMI', ',', 'Florida', '(',
↪ 'CNN', ')', '--', 'The', 'ninth', 'floor', 'of', 'the', 'Miami', '-', 'Dade',
↪'pretrial', 'detention', 'facility', 'is', 'dubbed', 'the', '"', 'forgotten', 'floor',
```

(continues on next page)

(continued from previous page)

```
```

First document in **CNN/DM** extractive **testing** set:

```
{'src': [['Marseille', ',', 'France', '(', 'CNN)The', 'French', 'prosecutor', 'leading',
↪'an', 'investigation', 'into', 'the', 'crash', 'of', 'Germanwings', 'Flight', '9525',
↪'insisted', 'Wednesday', 'that', 'he', 'was', 'not', 'aware', 'of', 'any', 'video',
↪'footage', 'from', 'on', 'board', 'the', 'plane', '.'], ['Marseille', 'prosecutor',
↪'Brice', 'Robin', 'told', 'CNN', 'that', '"', 'so', 'far', 'no', 'videos', 'were',
↪'used', 'in', 'the', 'crash', 'investigation', '.', '"'], ['He', 'added', ',', '"', 'A
↪', 'person', 'who', 'has', 'such', 'a', 'video', 'needs', 'to', 'immediately', 'give',
↪'it', 'to', 'the', 'investigators', '.', '"'], ['Robin', "'s", 'comments', 'follow',
↪'claims', 'by', 'two', 'magazines', ',', 'German', 'daily', 'Bild', 'and', 'French',
↪'Paris', 'Match', ',', 'of', 'a', 'cell', 'phone', 'video', 'showing', 'the',
↪'harrowing', 'final', 'seconds', 'from', 'on', 'board', 'Germanwings', 'Flight', '9525
↪', 'as', 'it', 'crashed', 'into', 'the', 'French', 'Alps', '.'], ['All', '150', 'on',
↪'board', 'were', 'killed', '.'], ['Paris', 'Match', 'and', 'Bild', 'reported', 'that',
↪'the', 'video', 'was', 'recovered', 'from', 'a', 'phone', 'at', 'the', 'wreckage',
↪'site', '.'], ['The', 'two', 'publications', 'described', 'the', 'supposed', 'video',
↪',', 'but', 'did', 'not', 'post', 'it', 'on', 'their', 'websites', '.'], ['The',
↪'publications', 'said', 'that', 'they', 'watched', 'the', 'video', ',', 'which', 'was',
↪ 'found', 'by', 'a', 'source', 'close', 'to', 'the', 'investigation', '.'], ['"', 'One
↪', 'can', 'hear', 'cries', 'of', "'", 'My', 'God', "'", 'in', 'several', 'languages',
↪',', '"', 'Paris', 'Match', 'reported', '.'], ['"', 'Metallic', 'banging', 'can', 'also
↪', 'be', 'heard', 'more', 'than', 'three', 'times', ',', 'perhaps', 'of', 'the', 'pilot
↪', 'trying', 'to', 'open', 'the', 'cockpit', 'door', 'with', 'a', 'heavy', 'object', '.
↪', ' '], ['Towards', 'the', 'end', ',', 'after', 'a', 'heavy', 'shake', ',', 'stronger
↪', 'than', 'the', 'others', ',', 'the', 'screaming', 'intensifies', '.'], ['"', 'It',
↪'is', 'a', 'very', 'disturbing', 'scene', ',', '"', 'said', 'Julian', 'Reichelt', ',',
↪'editor', '-', 'in', '-', 'chief', 'of', 'Bild', 'online', '.'], ['An', 'official',
↪'with', 'France', "'s", 'accident', 'investigation', 'agency', ',', 'the', 'BEA', ',',
↪'said', 'the', 'agency', 'is', 'not', 'aware', 'of', 'any', 'such', 'video', '.'], [
↪'Jean', '-', 'Marc', 'Menichini', ',', 'a', 'French', 'Gendarmerie', 'spokesman', 'in',
↪ 'charge', 'of', 'communications', 'on', 'rescue', 'efforts', 'around', 'the',
↪'Germanwings', 'crash', 'site', ',', 'told', 'CNN', 'that', 'the', 'reports', 'were', '
↪"', 'completely', 'wrong', '"', 'and', '"', 'unwarranted', '.', '"'], ['Cell', 'phones
↪', 'have', 'been', 'collected', 'at', 'the', 'site', ',', 'he', 'said', ',', 'but',
↪'that', 'they', '"', 'had', "n't", 'been', 'exploited', 'yet', '.'], ['Menichini',
↪'said', 'he', 'believed', 'the', 'cell', 'phones', 'would', 'need', 'to', 'be', 'sent',
↪ 'to', 'the', 'Criminal', 'Research', 'Institute', 'in', 'Rosny', 'sous', '-', 'Bois',
↪',', 'near', 'Paris', ',', 'in', 'order', 'to', 'be', 'analyzed', 'by', 'specialized',
↪'technicians', 'working', 'hand', '-', 'in', '-', 'hand', 'with', 'investigators', '.
↪'], ['But', 'none', 'of', 'the', 'cell', 'phones', 'found', 'so', 'far', 'have', 'been
↪', 'sent', 'to', 'the', 'institute', ',', 'Menichini', 'said', '.'], ['Asked', 'whether
↪', 'staff', 'involved', 'in', 'the', 'search', 'could', 'have', 'leaked', 'a', 'memory
↪', 'card', 'to', 'the', 'media', ',', 'Menichini', 'answered', 'with', 'a',
↪'categorical', '"', 'no', '.', '"'], ['Reichelt', 'told', '"', 'Erin', 'Burnett', ':'],
↪ ['Outfront', '"', 'that', 'he', 'had', 'watched', 'the', 'video', 'and', 'stood', 'by
↪', 'the', 'report', ',', 'saying', 'Bild', 'and', 'Paris', 'Match', 'are', '"', 'very',
↪ 'confident', '"', 'that', 'the', 'clip', 'is', 'real', '.'], ['He', 'noted', 'that',
↪'investigators', 'only', 'revealed', 'they', "'d", 'recovered', 'cell', 'phones', 'from
↪', 'the', 'crash', 'site', 'after', 'Bild', 'and', 'Paris', 'Match', 'published',
↪'their', 'reports', '.'], ['"', 'That', 'is', 'something', 'we', 'did', 'not', 'know',
↪'before', '.', '...'], ['Overall', 'we', 'can', 'say', 'many', 'things', 'of', 'the',
↪'investigation', 'were', "n't", 'revealed', 'by', 'the', 'investigation', 'at', 'the',
↪'beginning', ',', '"', 'he', 'said', '.'], ['German', 'airline', 'Lufthansa',
↪'confirmed', 'Tuesday', 'that', 'co', '-', 'pilot', 'Andreas', 'Lubitz', 'had',
↪'battled', 'depression', 'years', 'before', 'he', 'took', 'the', 'controls', 'of',
↪'Germanwings', 'Flight', '9525', ',', 'which', 'he', "'s", 'accused', 'of',
↪'deliberately', 'crashing', 'last', 'week', 'in', 'the', 'French', 'Alps', '.'], [
```

(continues on next page)

(continued from previous page)

```
```

8.6 Script Help

```
usage: convert_to_extractive.py [-h] [--base_output_path BASE_OUTPUT_PATH]
                                [--split_names {train,val,test} [{train,val,test} ...]]
                                [--add_target_to {train,val,test} [{train,val,test} ...]]
                                [--source_ext SOURCE_EXT] [--target_ext TARGET_EXT]
                                [--oracle_mode {greedy,combination}]
                                [--shard_interval SHARD_INTERVAL]
                                [--n_process N_PROCESS] [--batch_size BATCH_SIZE]
                                [--compression] [--resume]
                                [--tokenizer_log_interval TOKENIZER_LOG_INTERVAL]
                                [--sentencizer] [--no_preprocess]
                                [--min_sentence_ntokens MIN_SENTENCE_NTOKENS]
                                [--max_sentence_ntokens MAX_SENTENCE_NTOKENS]
                                [--min_example_nsents MIN_EXAMPLE_NSENTS]
                                [--max_example_nsents MAX_EXAMPLE_NSENTS]
                                [-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                                DIR

Convert an Abstractive Summarization Dataset to the Extractive Task

positional arguments:
DIR                   path to data directory

optional arguments:
-h, --help            show this help message and exit
--base_output_path BASE_OUTPUT_PATH
                        path to output processed data (default is `base_path`)
--split_names {train,val,test} [{train,val,test} ...]
                        which splits of dataset to process
--add_target_to {train,val,test} [{train,val,test} ...]
                        add the abstractive target to these splits (useful for
                        calculating rouge scores)
--source_ext SOURCE_EXT
                        extension of source files
--target_ext TARGET_EXT
                        extension of target files
--oracle_mode {greedy,combination}
                        method to convert abstractive summaries to extractive
                        summaries
--shard_interval SHARD_INTERVAL
                        how many examples to include in each shard of the dataset
                        (default: no shards)
--n_process N_PROCESS
                        number of processes for multithreading
--batch_size BATCH_SIZE
                        number of batches for tokenization
--compression         use gzip compression when saving data
```

(continues on next page)

(continued from previous page)

```
--resume              resume from last shard
--tokenizer_log_interval TOKENIZER_LOG_INTERVAL
                        minimum progress display update interval [default: 0.1]
                        seconds
--sentencizer         use a spacy sentencizer instead of a statistical model for
                        sentence detection (much faster but less accurate); see
                        https://spacy.io/api/sentencizer
--no_preprocess       do not run the preprocess function, which removes sentences
                        that are too long/short and examples that have too few/many
                        sentences
--min_sentence_ntokens MIN_SENTENCE_NTOKENS
                        minimum number of tokens per sentence
--max_sentence_ntokens MAX_SENTENCE_NTOKENS
                        maximum number of tokens per sentence
--min_example_nsents MIN_EXAMPLE_NSENTS
                        minimum number of sentences per example
--max_example_nsents MAX_EXAMPLE_NSENTS
                        maximum number of sentences per example
-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}, --log {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                        Set the logging level (default: 'Info').
```

CHAPTER NINE

AUTOMATIC PREPROCESSING

While the `convert_to_extractive.py` script prepares a dataset for the extractive task, the data still needs to be processed for usage with a machine learning model. This preprocessing depends on the chosen model, and thus is implemented in the `extractive.py` file with the rest of the model training logic.

The actual *`ExtractiveSummarizer`* LightningModule (which is similar to an `nn.Module` but with a built-in training loop, more info at the pytorch_lightning documentation) implements a *`prepare_data()`* function. This function is automatically called by `pytorch_lightning` to load and process the examples.

Note: Memory Usage Note: If sharding was turned off during the `convert_to_extractive` process then *`prepare_data()`* will run once, loading the entire dataset into memory to process just like the `convert_to_extractive.py` script.

There is a `--only_preprocess` argument available to only run the pre-process step and exit the script after all the examples have been written to disk. This option will force data to be preprocessed, even if it was already computed and is detected on disk, and any previous processed files will be overwritten.

Thus, to only pre-process data for use when training a model run:

```
python main.py --data_path ./datasets/cnn_dailymail_processor/cnn_dm --use_logger␣
↪tensorboard --model_name_or_path bert-base-uncased --model_type bert --do_train --only_
↪preprocess
```

Warning: If processed files are detected, they will automatically be loaded from disk. This includes any files that follow the pattern `[dataset_split_name].*.pt`, where * is any text of any length.

CHAPTER

TEN

TRAINING AN EXTRACTIVE SUMMARIZATION MODEL

10.1 Details

Once the dataset has been converted to the extractive task, it can be used as input to a *data.SentencesProcessor*, which has a *add_examples()* function to add sets of `(example, labels)` and a *get_features()* function that processes the data and prepares it to be inputted into the model (`input_ids`, `attention_masks`, `labels`, `token_type_ids`, `sent_rep_token_ids`, `sent_rep_token_ids_masks`). Feature extraction runs in parallel and tokenizes text using the tokenizer appropriate for the model specified with `--model_name_or_path`. The tokenizer can be changed to another `huggingface/transformers` tokenizer with the `--tokenizer_name` option.

Important: When loading a pre-trained model you may encounter this common error:

```
RuntimeError: Error(s) in loading state_dict for ExtractiveSummarizer:
Missing key(s) in state_dict: "word_embedding_model.embeddings.position_ids".
```

To solve this issue, set `strict=False` like so: `model = ExtractiveSummarizer.load_from_checkpoint("distilroberta-base-ext-sum.ckpt", strict=False)`. If you are using the `main.py` script, then you can alternatively sepcify the `--no_strict` option.

For the *CNN/DM dataset*, to train a model for 50,000 steps on the data run:

```
python main.py --data_path ./datasets/cnn_dailymail_processor/cnn_dm --weights_save_path␣
↪./trained_models --do_train --max_steps 50000
```

- The `--do_train` argument runs the training process. Set *–do_test* to test after training.
- The `--data_path` argument specifies where the extractive dataset json file are located.
- The *–weights_save_path* argument specifies where the model weights should be stored.

If you prefer to measure training progress by epochs instead of steps, you can use the `--max_epochs` and `--min_epochs` options.

The batch size can be changed with the `--batch_size` option. This changes the batch size for training, validation, and testing. You can set the `--auto_scale_batch_size` option to automatically determine this value. See "Auto scaling of batch size" from the pytorch_lightning documentation for more information about the algorithm and available options.

If the extractive dataset json files are compressed using gzip, then they will be automatically decompressed during the data preprocessing step of training.

By default, the model weights are saved after every epoch to the `--default_root_dir`. The logs are also saved to this folder. You can change the weight save path (separate folder for logs and weights) with the `--weights_save_path` option.

The length of output summaries during testing is 3 by default. You can change this by setting the `--test_k` option to the number of sentences desired in generated summaries. This assumes `--test_id_method` is set to `top_k`, which is the default. `top_k` selects the top `k` sentences and the other option, `greater_k`, selects those sentences with a rank above `k`. `k` is specified by the `--test_k` argument.

Important: More example training commands can be found on the TransformerSum Weights & Biases page. Just click the name of a training run, go to the overview page by clicking the "i" icon in the top left, and look at the command value.

10.2 Pooling Modes

The pooling model determines how word vectors should be converted to sentence embeddings. The implementation can be found in *pooling.py*. The `--pooling_mode` argument can be set to either `sent_rep_tokens` or `mean_tokens`. While the pooling `nn.Module` allows multiple methods to be used at once (it will concatenate and return the results), the training script does not.

- `sent_rep_tokens`: Uses the sentence representation token (commonly called the classification token; `[CLS]` in BERT and <s> in RoBERTa) vectors as sentence embeddings.
- `mean_tokens`: Uses the average of the token vectors for each sentence in the input as sentence embeddings.
- `max_tokens`: Uses the maximum of the token vectors for each sentence in the input as sentence embeddings.

10.3 Custom Models

You can use any autoencoding transformer model for the word embedding model (by setting the `--model_name_or_path` CLI argument) as long as it was saved in the `huggingface/transformers` format. Any model that is loaded with this option by specifying a path is considered "custom" in this project. Currently, there are no "custom" models that are "officially" supported. The *longformer* used to be a custom model, but it was since added to the *huggingface/transformers* repository, and thus can be used in this project just like any other model.

10.4 Script Help

Output of `python main.py --mode extractive --help` (*generic options* removed):

```
usage: main.py [-h]
            [--model_name_or_path MODEL_NAME_OR_PATH] [--model_type MODEL_TYPE]
            [--tokenizer_name TOKENIZER_NAME] [--tokenizer_no_use_fast]
            [--max_seq_length MAX_SEQ_LENGTH] [--data_path DATA_PATH]
            [--data_type {txt,pt,none}] [--num_threads NUM_THREADS]
            [--processing_num_threads PROCESSING_NUM_THREADS]
            [--pooling_mode {sent_rep_tokens,mean_tokens,max_tokens}]
            [--num_frozen_steps NUM_FROZEN_STEPS] [--batch_size BATCH_SIZE]
            [--dataloader_type {map,iterable}]
            [--dataloader_num_workers DATALOADER_NUM_WORKERS]
            [--processor_no_bert_compatible_cls] [--only_preprocess]
            [--preprocess_resume] [--create_token_type_ids {binary,sequential}]
            [--no_use_token_type_ids]
```

(continues on next page)

(continued from previous page)

```
            [--classifier {linear,simple_linear,transformer,transformer_linear}]
            [--classifier_dropout CLASSIFIER_DROPOUT]
            [--classifier_transformer_num_layers CLASSIFIER_TRANSFORMER_NUM_LAYERS]
            [--train_name TRAIN_NAME] [--val_name VAL_NAME]
            [--test_name TEST_NAME] [--test_id_method {greater_k,top_k}]
            [--test_k TEST_K] [--no_test_block_trigrams] [--test_use_pyrouge]
            [--loss_key {loss_total,loss_total_norm_batch,loss_avg_seq_sum,loss_avg_seq_
↪mean,loss_avg}]

optional arguments:
-h, --help            show this help message and exit
--model_name_or_path MODEL_NAME_OR_PATH
                        Path to pre-trained model or shortcut name. A list of
                        shortcut names can be found at https://huggingface.co/tran
                        sformers/pretrained_models.html. Community-uploaded models
                        are located at https://huggingface.co/models.
--model_type MODEL_TYPE
                        Model type selected in the list: retribert, t5,
                        distilbert, albert, camembert, xlm-roberta, bart,
                        longformer, roberta, bert, openai-gpt, gpt2, mobilebert,
                        transfo-xl, xlnet, flaubert, xlm, ctrl, electra, reformer
--tokenizer_name TOKENIZER_NAME
--tokenizer_no_use_fast
                        Don't use the fast version of the tokenizer for the
                        specified model. More info: https://huggingface.co/transfo
                        rmers/main_classes/tokenizer.html.
--max_seq_length MAX_SEQ_LENGTH
                        The maximum sequence length of processed documents.
--data_path DATA_PATH
                        Directory containing the dataset.
--data_type {txt,pt,none}
                        The file extension of the prepared data. The 'map'
                        `--dataloader_type` requires `txt` and the 'iterable'
                        `--dataloader_type` works with both. If the data is not
                        prepared yet (in JSON format) this value specifies the
                        output format after processing. If the data is prepared,
                        this value specifies the format to load. If it is `none`
                        then the type of data to be loaded will be inferred from
                        the `data_path`. If data needs to be prepared, this cannot
                        be set to `none`.
--num_threads NUM_THREADS
--processing_num_threads PROCESSING_NUM_THREADS
--pooling_mode {sent_rep_tokens,mean_tokens,max_tokens}
                        How word vectors should be converted to sentence
                        embeddings.
--num_frozen_steps NUM_FROZEN_STEPS
                        Freeze (don't train) the word embedding model for this
                        many steps.
--batch_size BATCH_SIZE
                        Batch size per GPU/CPU for training/evaluation/testing.
--dataloader_type {map,iterable}
                        The style of dataloader to use. `map` is faster and uses
```

(continues on next page)

(continued from previous page)

```
                        less memory.
--dataloader_num_workers DATALOADER_NUM_WORKERS
                        The number of workers to use when loading data. A general
                        place to start is to set num_workers equal to the number
                        of CPU cores on your machine. If `--dataloader_type` is
                        'iterable' then this setting has no effect and num_workers
                        will be 1. More details here: https://pytorch-
                        lightning.readthedocs.io/en/latest/performance.html#num-
                        workers
--processor_no_bert_compatible_cls
                        If model uses bert compatible [CLS] tokens for sentence
                        representations.
--only_preprocess     Only preprocess and write the data to disk. Don't train
                        model. This will force data to be preprocessed, even if it
                        was already computed and is detected on disk, and any
                        previous processed files will be overwritten.
--preprocess_resume   Resume preprocessing. `--only_preprocess` must be set in
                        order to resume. Determines which files to process by
                        finding the shards that do not have a coresponding ".pt"
                        file in the data directory.
--create_token_type_ids {binary,sequential}
                        Create token type ids during preprocessing.
--no_use_token_type_ids
                        Set to not train with `token_type_ids` (don't pass them
                        into the model).
--classifier {linear,simple_linear,transformer,transformer_linear}
                        Which classifier/encoder to use to reduce the hidden
                        dimension of the sentence vectors. `linear` - a
                        `LinearClassifier` with two linear layers, dropout, and an
                        activation function. `simple_linear` - a
                        `LinearClassifier` with one linear layer and a sigmoid.
                        `transformer` - a `TransformerEncoderClassifier` which
                        runs the sentence vectors through some
                        `nn.TransformerEncoderLayer`s and then a simple
                        `nn.Linear` layer. `transformer_linear` - a
                        `TransformerEncoderClassifier` with a `LinearClassifier`
                        as the `reduction` parameter, which results in the same
                        thing as the `transformer` option but with a
                        `LinearClassifier` instead of a `nn.Linear` layer.
--classifier_dropout CLASSIFIER_DROPOUT
                        The value for the dropout layers in the classifier.
--classifier_transformer_num_layers CLASSIFIER_TRANSFORMER_NUM_LAYERS
                        The number of layers for the `transformer` classifier.
                        Only has an effect if `--classifier` contains
                        "transformer".
--train_name TRAIN_NAME
                        name for set of training files on disk (for loading and
                        saving)
--val_name VAL_NAME   name for set of validation files on disk (for loading and
                        saving)
--test_name TEST_NAME
                        name for set of testing files on disk (for loading and
```

(continues on next page)

(continued from previous page)

```
                        saving)
--test_id_method {greater_k,top_k}
                        How to chose the top predictions from the model for ROUGE
                        scores.
--test_k TEST_K       The `k` parameter for the `--test_id_method`. Must be set
                        if using the `greater_k` option. (default: 3)
--no_test_block_trigrams
                        Disable trigram blocking when calculating ROUGE scores
                        during testing. This will increase repetition and thus
                        decrease accuracy.
--test_use_pyrouge    Use `pyrouge`, which is an interface to the official ROUGE
                        software, instead of the pure-python implementation
                        provided by `rouge-score`. You must have the real ROUGE
                        package installed. More details about ROUGE 1.5.5 here: ht
                        tps://github.com/andersjo/pyrouge/tree/master/tools/ROUGE-
                        1.5.5. It is recommended to use this option for official
                        scores. The `ROUGE-L` measurements from `pyrouge` are
                        equivalent to the `rougeLsum` measurements from the
                        default `rouge-score` package.
--loss_key {loss_total,loss_total_norm_batch,loss_avg_seq_sum,loss_avg_seq_mean,loss_avg}
                        Which reduction method to use with BCELoss. See the
                        `experiments/loss_functions/` folder for info on how the
                        default (`loss_avg_seq_mean`) was chosen.
```

CHAPTER

ELEVEN

EXPERIMENTS

Interactive charts, graphs, raw data, run commands, hyperparameter choices, and more for all experiments are publicly available on the TransformerSum Weights & Biases page. You can download the raw data for each model on this site, or download an overview as a CSV. Please open an issue if you have questions about these experiments.

Important notes when running experiments:

- If you are using `--overfit_batches`, then `overfit_batches` percent of the testing data is being used as well as `overfit_batches` percent of the training data. Due to the way `pytorch_lightning` was written, it is necessary to use the same `batch_size` when using `overfit_batches` in order to get the exact same results. I currently am not sure why this is the case but removing `overfit_batches` and using different `batch_sizes` produces identical results. Open an issue or submit a pull request if you know why.
- Have another note that should be stated here? Open an issue. All contributions are very helpful.

The *Version 1* were conducted on a previous version of `TransformerSum` that contained bugs. Thus, the scores and graphs of the older experiments don't represent model performance but their results relative to each other should still be accurate. The *Version 2* were conducted on a new version without bugs and thus should be easily reproducible.

11.1 Version 3

All version 3 extractive models were trained for three epochs with gradient accumulation every two steps. The AdamW optimizer was used with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e{-8}$. Models were trained on one NVIDIA Tesla P100 GPU. Unless otherwise specified, the learning rate was 2e-5 and a linear warmup with decay learning rate scheduler was used with 1400 steps of warmup (technically 2800 steps due to gradient accumulation every two steps). Except during their respective experiments, the `simple_linear` classifier and `sent_rep_tokens` pooling method are used. Gradients are clipped at 1.0 during training for all models. Model checkpoints were saved and evaluated on the validation set at the end of each epoch. ROUGE scores are reported on the test set of the model checkpoint saved after the final epoch.

11.1.1 Pooling Mode

Wandb Tag: `pooling-mode-test-v3`

All three pooling modes (`mean_tokens`, `sentence_rep_tokens`, and `max_tokens`) were tested using DistilBERT and DistilRoBERTa, which are warm started from the `distilbert-base-uncased` and `distilroberta-base` checkpoints, respectively. I only test the `distil*` models since they reach at least 95% of the performance of the original model while being significantly faster to train. The models were trained and tested on CNN/DailyMail, WikiHow, and ArXiv/PubMed to determine if certain methods worked better with certain topics. All models were trained with a batch size of 32 and the hyperparameters discussed in above at *Version 3*.

Pooling Mode Results

ROUGE Scores:

Model	Pooling Method	CNN/DM	WikiHow	ArXiv/PubMed
distilbert-base-uncased	sent_rep	42.71/19.91/39.18	30.69/08.65/28.58	34.93/12.21/31.00
	mean	42.70/19.88/39.16	30.48/08.56/28.42	34.48/11.89/30.61
	max	42.74/19.90/39.17	30.51/08.62/28.43	34.50/11.91/30.62
distilroberta-base	sent_rep	42.87/20.02/39.31	31.07/08.96/28.95	34.70/12.16/30.82
	mean	43.00/20.08/39.42	30.96/08.93/28.86	34.24/11.82/30.42
	max	42.91/20.04/39.33	30.93/08.92/28.82	34.28/11.82/30.44

Main Takeaway: Across all datasets and models, the pooling mode has no significant impact on the final ROUGE scores. However, the sent_rep method usually performs slightly better. Additionally, the mean and max methods are about 30% slower than the sent_rep pooling method.

11.1.2 Classifier/Encoder

Wandb Tag: `encoder-test-v3`

All four summarization layers, including two variations of the verb|transformer| method for a total of five configurations, were tested using the same models and datasets from the *pooling modes experiment*. For this experiment, a batch size of 32 was used.

Classifier/Encoder Results

ROUGE Scores:

Model	Classifier	CNN/DM	WikiHow	ArXiv/PubMed
distilbert-base-uncased	simple_linear	42.71/19.91/39.18	30.69/08.65/28.58	34.93/12.21/31.00
	linear	42.70/19.84/39.14	30.67/08.62/28.56	34.87/12.20/30.96
	transformer	42.78/19.93/39.22	30.66/08.69/28.57	34.94/12.22/31.03
	transformer_linear	42.78/19.93/39.22	30.64/08.64/28.54	34.97/12.22/31.02
distilroberta-base	simple_linear	42.87/20.02/39.31	31.07/08.96/28.95	34.70/12.16/30.82
	linear	43.18/20.26/39.61	31.08/08.98/28.96	34.77/12.17/30.88
	transformer	42.94/20.03/39.37	31.05/08.97/28.93	34.77/12.17/30.87
	transformer_linear	42.90/20.00/39.34	31.13/09.01/29.02	34.77/12.18/30.88

Main Takeaway: There is no significant difference between the classifier used. Thus, you should use the linear classifier by default since it contains fewer parameters.

11.2 Version 2

11.2.1 Classifier/Encoder `simple_linear` vs `linear`

Commit *dfefd15* added a `SimpleLinearClassifier`. This experiment servers to determine if `simple_linear` (`SimpleLinearClassifier`) is better than `linear` (`LinearClassifier`).

Command used to run the tests:

```
python main.py \
--model_name_or_path distilbert-base-uncased \
--model_type distilbert \
--no_use_token_type_ids \
--use_custom_checkpoint_callback \
--data_path ./pt/bert-base-uncased \
--max_epochs 3 \
--accumulate_grad_batches 2 \
--warmup_steps 1400 \
--gradient_clip_val 1.0 \
--optimizer_type adamw \
--use_scheduler linear \
--do_train --do_test \
--batch_size 32 \
--classifier [`linear` or `simple_linear`]
```

Classifier/Encoder Results

Training Times and Model Sizes:

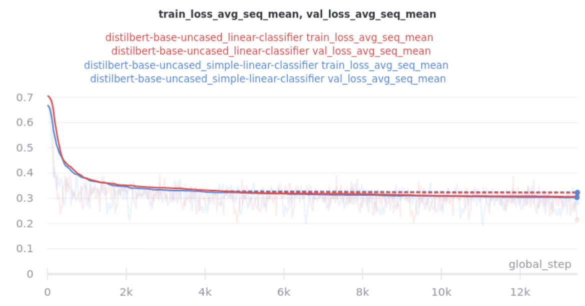
Model Key	Time	Model Size
`linear`	6h 28m 21s	810.6MB
`simple_linear`	6h 22m 32s	796.4MB

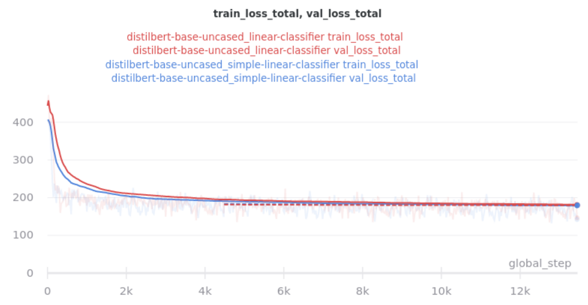
ROUGE Scores:

Name	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-L-Sum
`linear`	42.8	19.9	27.5	39.2
`simple_linear`	42.7	19.9	27.5	39.2

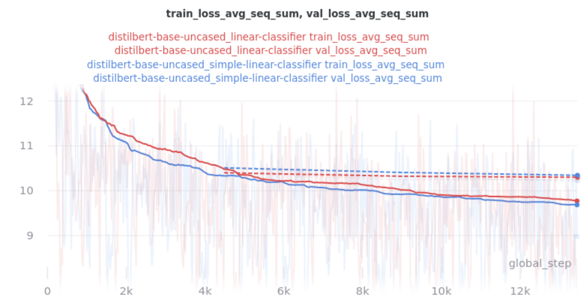
Main Takeaway: There is no significant difference in performance between the `linear` and `simple_linear` classifiers/encoders. However, `simple_linear` is better due to its lower training and validation loss.

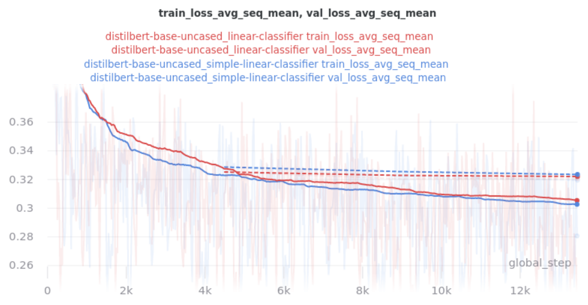
Outliers Included:

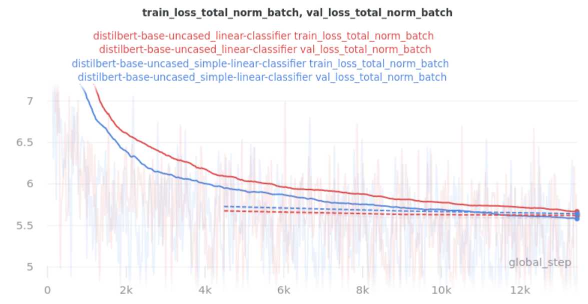


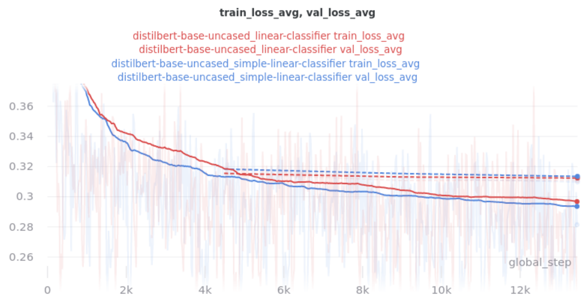

train_loss_total, val_loss_total
distilbert-base-uncased_linear-classifier train_loss_total
distilbert-base-uncased_linear-classifier val_loss_total
distilbert-base-uncased_simple-linear-classifier train_loss_total
distilbert-base-uncased_simple-linear-classifier val_loss_total
global_step

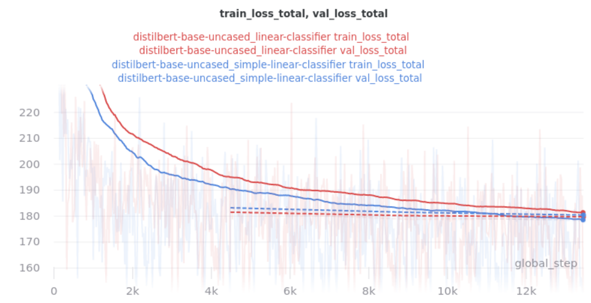
No Outliers:

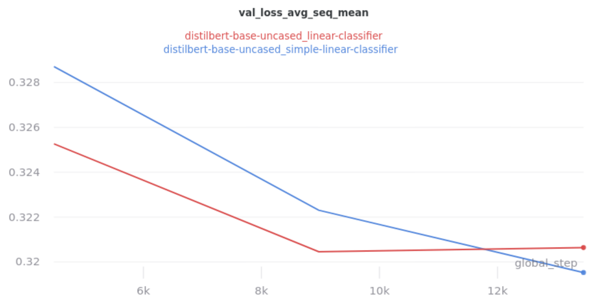

train_loss_avg_seq_sum, val_loss_avg_seq_sum
distilbert-base-uncased_linear-classifier train_loss_avg_seq_sum
distilbert-base-uncased_linear-classifier val_loss_avg_seq_sum
distilbert-base-uncased_simple-linear-classifier train_loss_avg_seq_sum
distilbert-base-uncased_simple-linear-classifier val_loss_avg_seq_sum
global_step


train_loss_avg_seq_mean, val_loss_avg_seq_mean
distilbert-base-uncased_linear-classifier train_loss_avg_seq_mean
distilbert-base-uncased_linear-classifier val_loss_avg_seq_mean
distilbert-base-uncased_simple-linear-classifier train_loss_avg_seq_mean
distilbert-base-uncased_simple-linear-classifier val_loss_avg_seq_mean
global_step


train_loss_total_norm_batch, val_loss_total_norm_batch
distilbert-base-uncased_linear-classifier train_loss_total_norm_batch
distilbert-base-uncased_linear-classifier val_loss_total_norm_batch
distilbert-base-uncased_simple-linear-classifier train_loss_total_norm_batch
distilbert-base-uncased_simple-linear-classifier val_loss_total_norm_batch
global_step


train_loss_avg, val_loss_avg
distilbert-base-uncased_linear-classifier train_loss_avg
distilbert-base-uncased_linear-classifier val_loss_avg
distilbert-base-uncased_simple-linear-classifier train_loss_avg
distilbert-base-uncased_simple-linear-classifier val_loss_avg
global_step





11.3 Version 1

Important: These experiments may be difficult to reproduce because they were conducted on an early version of the project that contained several bugs.

Reproducibility Notes:

Bugs present in the version these experiments were conducted with:

1. Sentences were not split properly when computing ROUGE scores (fixed in commit dfefd15).
2. Data was missing from the training, validation, and testing sets (fixed in commit 4de5532).
3. Tokens were not converted to lowercase for models with the word "uncased" in their name (fixed in commit d934e09).
4. `rougeLsum` is not reported. See *Extractive vs Abstractive Summarization* for the difference between `rougeL` and `rougeLsum` (fixed in commit d934e09).
5. Trigram blocking was not used (fixed in commit 60f868e).

Despite these differences from the official models, the relative results of these experiments should hold true, so their general findings should remain constant with newer models. If you find conflicting results please open an issue.

11.3.1 Loss Functions

The loss function implementation can be found in the *extractive.ExtractiveSummarizer.compute_loss()* function. The function uses `nn.BCELoss` with `reduction="none"` and then applies 5 different reduction techniques. Special reduction methods were needed to ignore padding and operate on the multi-class-per-document approach (each input is assigned more than one of the same class) that this research uses to perform extractive summarization. See the comments throughout the function for more information. The five different reduction methods were tested with the `distilbert-base-uncased` word embedding model and the `pooling_mode` set to `sent_rep_tokens`. Training time is just under 4 hours on a Tesla P100 (3h52m average).

The `--loss_key` argument specifies the reduction method to use. It can be one of the following: `loss_total`, `loss_total_norm_batch`, `loss_avg_seq_sum`, `loss_avg_seq_mean`, `loss_avg`.

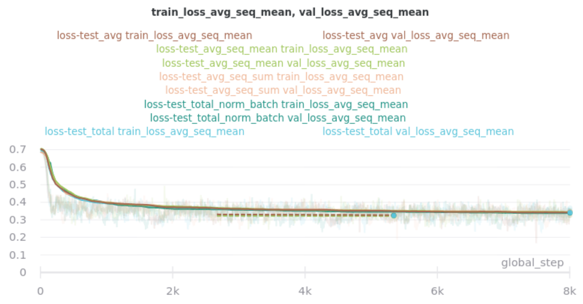
Full command used to run the tests:

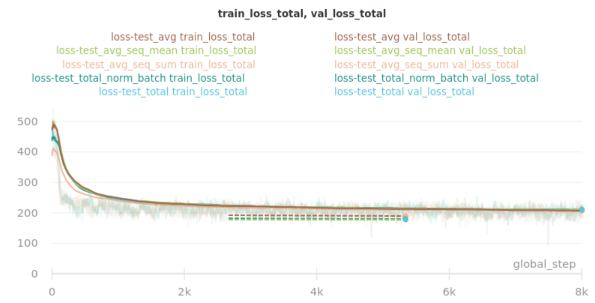
```
python main.py \
--model_name_or_path distilbert-base-uncased \
--no_use_token_type_ids \
--pooling_mode sent_rep_tokens \
--data_path ./cnn_dm_pt/bert-base-uncased \
--max_epochs 3 \
--accumulate_grad_batches 2 \
--warmup_steps 1800 \
--overfit_batches 0.6 \
--gradient_clip_val 1.0 \
--optimizer_type adamw \
--use_scheduler linear \
--profiler \
--do_train --do_test \
--loss_key [Loss Key Here] \
--batch_size 32
```

Loss Functions Results

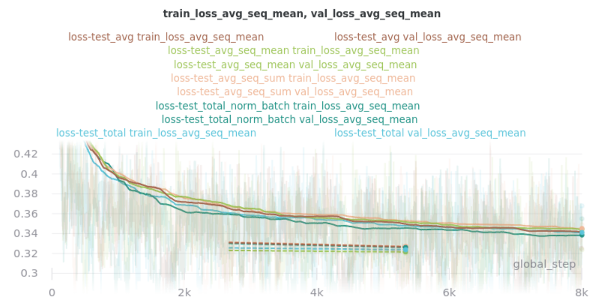
Graph Legend Description: The `loss-test` label (the first part) is the experiment, which indicates the loss reduction method that was tested. The second part of each key is the graphed quantity. For example, the first line of the key for the first graph in the `Outliers Included` section below indicates that `loss_avg` was tested and that its results as measured by the `loss_avg_seq_mean` reduction method are shown in brown. The train results are solid brown and the validation results are dotted brown.

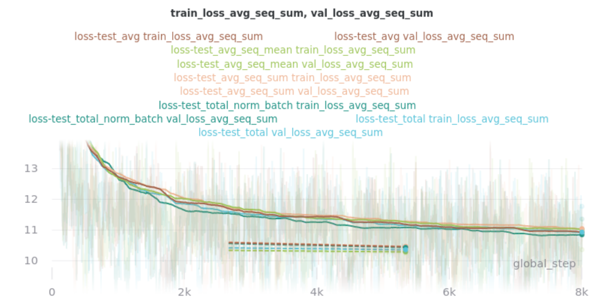
Outliers Included:

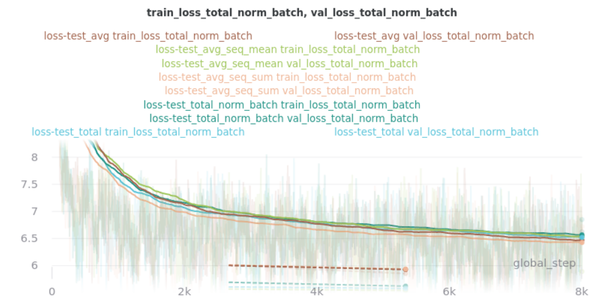


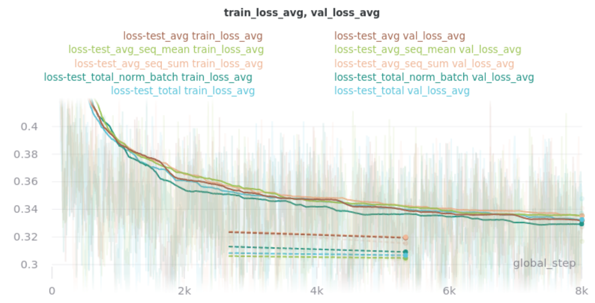

train_loss_total, val_loss_total
loss-test_avg train_loss_total
loss-test_avg_seq_mean train_loss_total
loss-test_avg_seq_sum train_loss_total
loss-test_total_norm_batch train_loss_total
loss-test_total train_loss_total
loss-test_avg val_loss_total
loss-test_avg_seq_mean val_loss_total
loss-test_avg_seq_sum val_loss_total
loss-test_total_norm_batch val_loss_total
loss-test_total val_loss_total
global_step

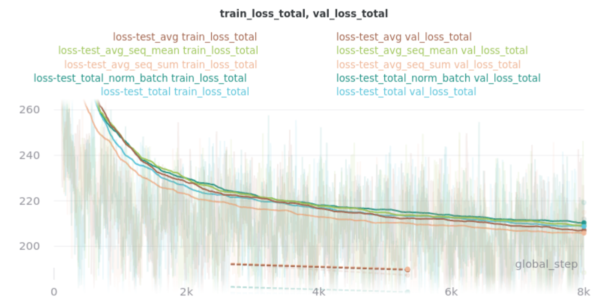
No Outliers:

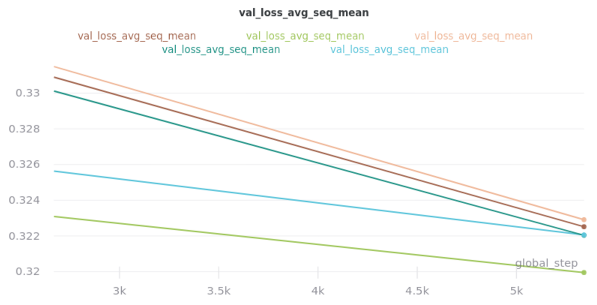

train_loss_avg_seq_mean, val_loss_avg_seq_mean
global_step


train_loss_avg_seq_sum, val_loss_avg_seq_sum
global_step


train_loss_total_norm_batch, val_loss_total_norm_batch
global_step


train_loss_avg, val_loss_avg
global_step





The CSV files the were used to generate the above graphs can be found in `experiments/loss_functions`.

Based on the results, `loss_avg_seq_mean` was chosen as the default.

11.3.2 Word Embedding Models

Different transformer models of various architectures and sizes were tested.

Tested Models:

Model Type	Model Key	Batch Size
Distil*	`distilbert-base-uncased`, `distilroberta-base`	16
Base	`bert-base-uncased`, `roberta-base`, `albert-base-v2`	16
Large	`bert-large-uncased`, `roberta-large`, `albert-xlarge-v2`	4

Albert Info: The above batch sizes are true except for `albert` models, which have special batch sizes due to the increased memory needed to train them*. *``albert-base-v2`` was trained with a batch size of ``12`` and ``albert-xlarge-v2`` with a batch size of ``2``.*

Model	Parameters	Layers	Hidden	Heads	Embedding	Parameter-sharing
BERT-base	110M	12	768	12	768	False
BERT-large	340M	24	1024	16	1024	False
ALBERT-base	12M	12	768	12	128	True
ALBERT-large	18M	24	1024	16	128	True
ALBERT-xlarge	59M	24	2048	32	128	True
ALBERT-xxlarge	233M	12	4096	64	128	True

*The huggingface/transformers documentation says “ALBERT uses repeating layers which results in a small memory footprint.” This may be true but I found that the normal batch sizes I used for the base and large models would crash the training script when `albert` models were used. Thus, the batch sizes were decreased. The advantage that of `albert` that I found was incredibly small model weight checkpoint files (see results below for sizes).

All models were trained for 3 epochs (except albert-xlarge-v2) (which will result in different numbers of steps but will ensure that each model saw the same amount of information), using the AdamW optimizer with a linear scheduler with 1800 steps of warmup. Gradients were accumulated every 2 batches and clipped at 1.0. **Only 60% of the data was used** (to decrease training time, but also will provide similar results if all the data was used). --no_use_token_type_ids was set if the model was not compatible with token type ids.

Full command used to run the tests:

```
python main.py \
--model_name_or_path [Model Name] \
--model_type [Model Type] \
--pooling_mode sent_rep_tokens \
--data_path ./cnn_dm_pt/[Model Type]-base \
--max_epochs 3 \
--accumulate_grad_batches 2 \
--warmup_steps 1800 \
--overfit_batches 0.6 \
--gradient_clip_val 1.0 \
--optimizer_type adamw \
--use_scheduler linear \
--profiler \
--do_train --do_test \
--batch_size [Batch Size]
```

WEB Results

The CSV files the were used to generate the below graphs can be found in experiments/web.

All ROUGE Scores are test set results on the CNN/DailyMail dataset using ROUGE F1.

All model sizes are not compressed. They are the raw .ckpt output file sizes of the best performing epoch by val_loss.

Final (Combined) Results

The loss_total, loss_avg_seq_sum, and loss_total_norm_batch loss reduction techniques depend on the batch size. That is, the larger the batch size, the larger these losses will be. The loss_avg_seq_mean and loss_avg do not depend on the batch size since they are averages instead of totals. Therefore, only the non-batch-size-dependent metrics were used for the final results because difference batch sizes were used.

Distil* Models

More information about distil* models found in the huggingface/transformers examples.

Warning: Distil* models do not accept token type ids. So set --no_use_token_type_ids while training using the above command.

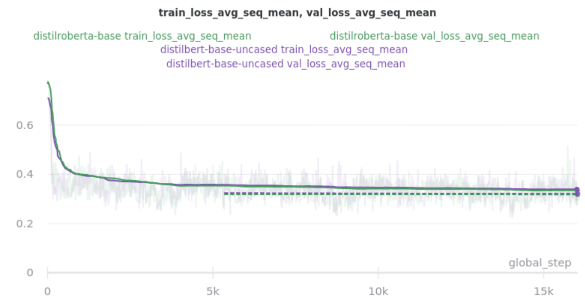
Training Times and Model Sizes:

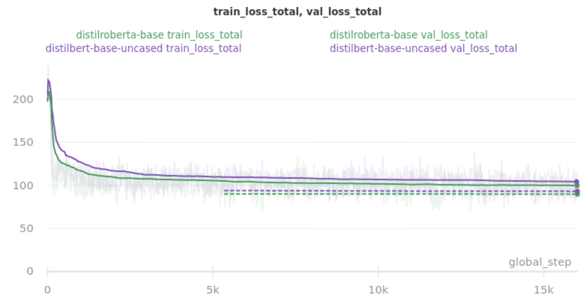
Model Key	Time	Model Size
distilbert-base-uncased	4h 5m 30s	810.6MB
distilroberta-base	4h 12m 53s	995.0MB

ROUGE Scores:

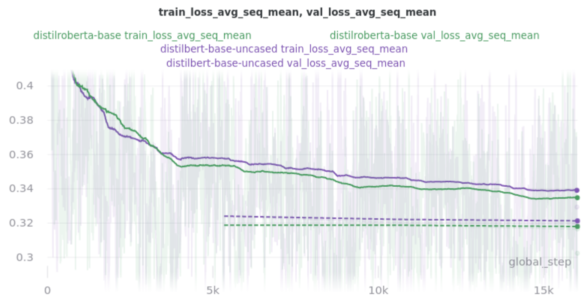
Name	ROUGE-1	ROUGE-2	ROUGE-L
distilbert-base-uncased	40.1	18.1	26.0
distilroberta-base	40.9	18.7	26.4

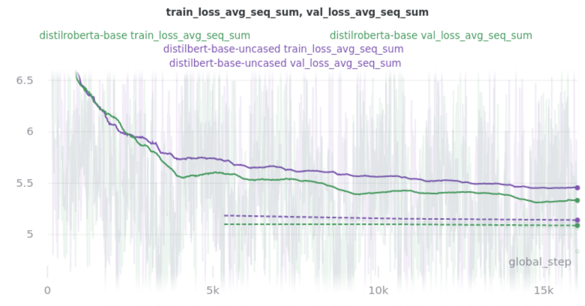
Outliers Included:

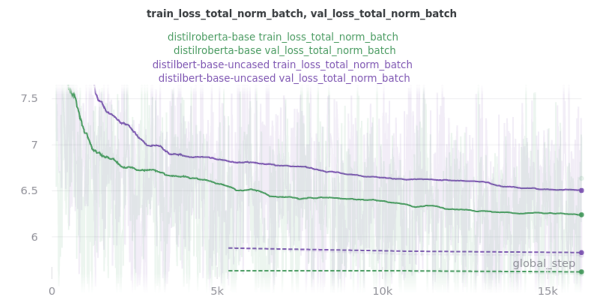


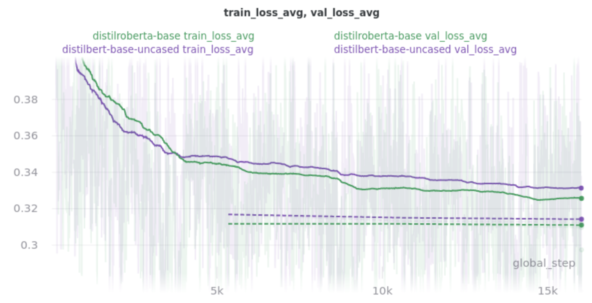


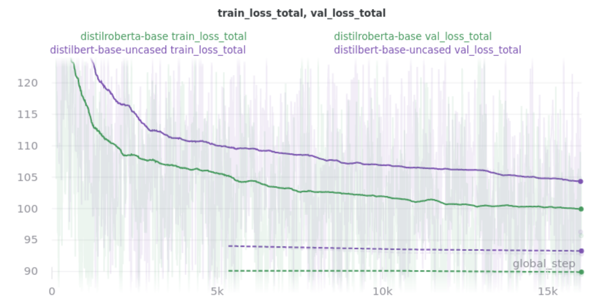
No Outliers:

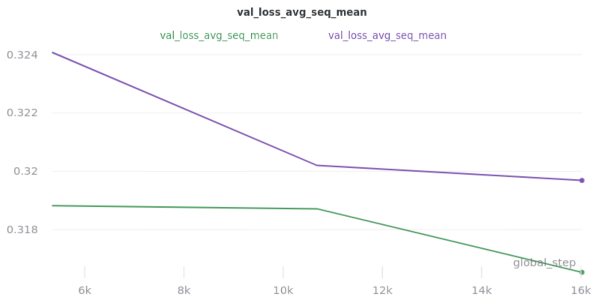












Base Models

Warning: `roberta-base` does not accept token type ids. So set `--no_use_token_type_ids` while training using the above command.

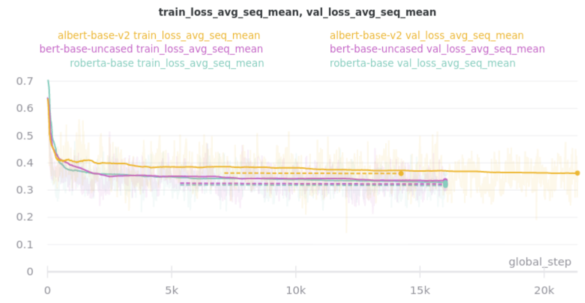
Training Times and Model Sizes:

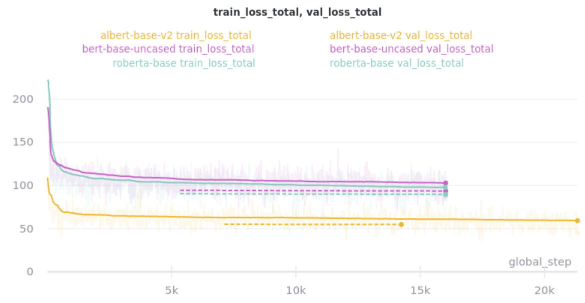
Model Key	Time	Model Size
`bert-base-uncased`	7h 56m 39s	1.3GB
`roberta-base`	7h 52m 0s	1.5GB
`albert-base-v2`	7h 32m 19s	149.7MB

ROUGE Scores:

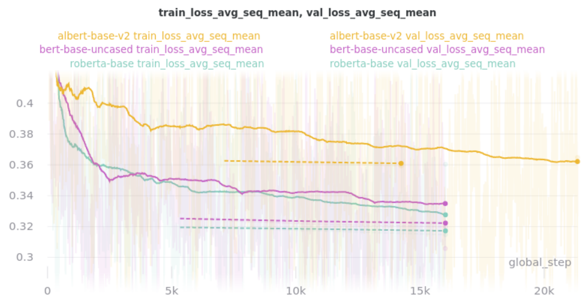
Name	ROUGE-1	ROUGE-2	ROUGE-L
bert-base-uncased	40.2	18.2	26.1
roberta-base	42.3	20.1	27.4
albert-base-v2	40.5	18.4	26.1

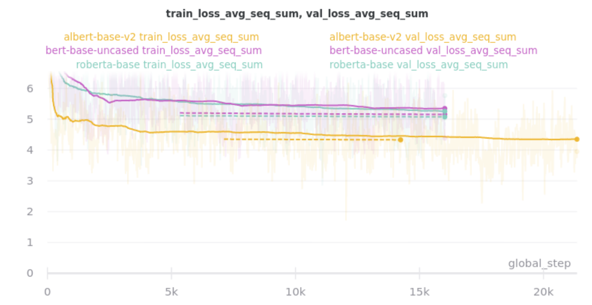
Outliers Included:

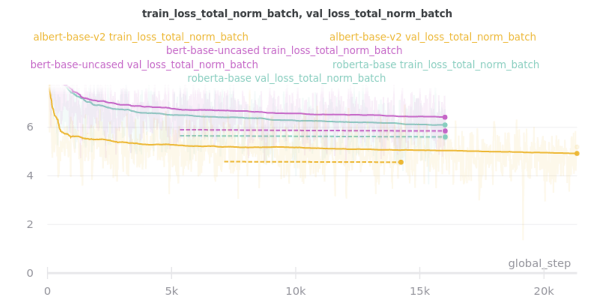


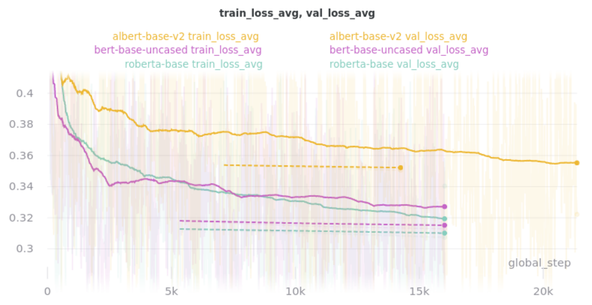


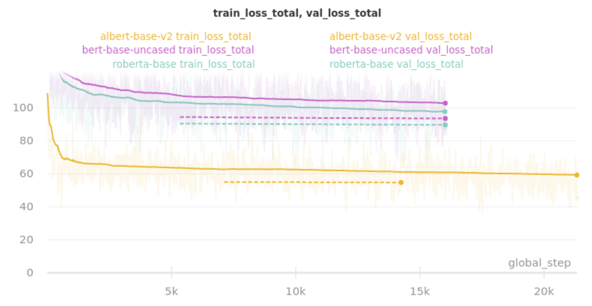
No Outliers:

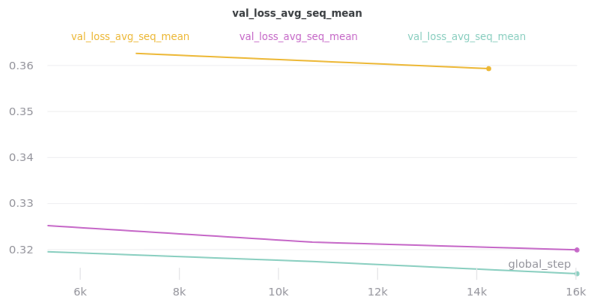






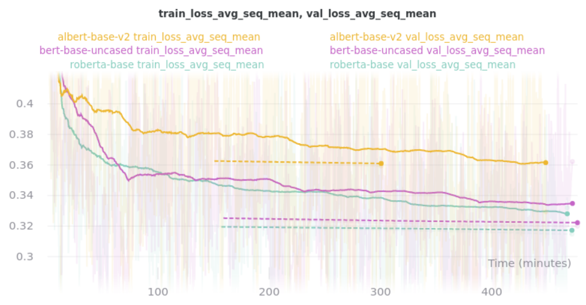






Relative Time:

This is included because the batch size for `albert-base-v2` had to be lowered to 12 (from 16).



Large Models

Warning: `roberta-large` does not accept token type ids. So set `--no_use_token_type_ids` while training using the above command.

Important: `albert-xlarge-v2` (batch size 2) was set to be trained with for 2 epochs instead of 3, but was stopped early at `global_step` 56394.

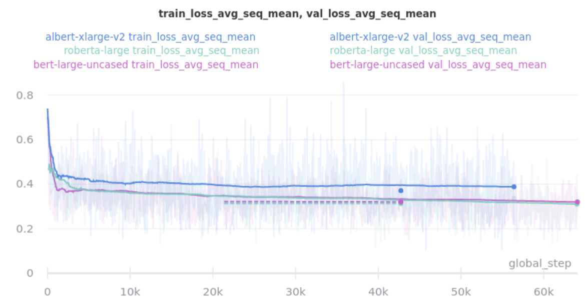
Training Times and Model Sizes:

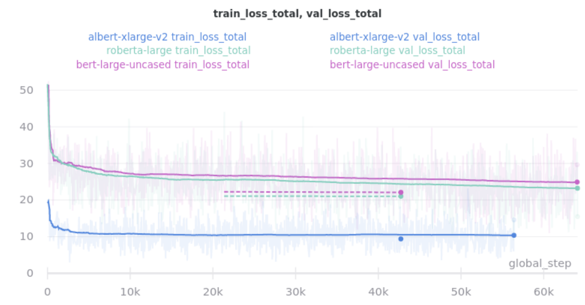
Model Key	Time	Model Size
`bert-large-uncased`	17h 55m 18s	4.0GB
`roberta-large`	18h 32m 28s	4.3GB
`albert-xlarge-v2`	21h 15m 54s	708.9MB

ROUGE Scores:

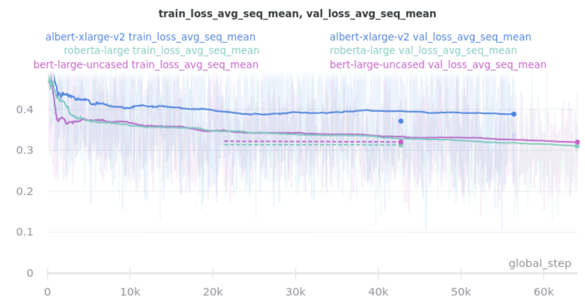
Name	ROUGE-1	ROUGE-2	ROUGE-L
bert-large-uncased	41.5	19.3	27.0
roberta-large	41.5	19.3	27.0
albert-xlarge-v2	40.7	18.4	26.1

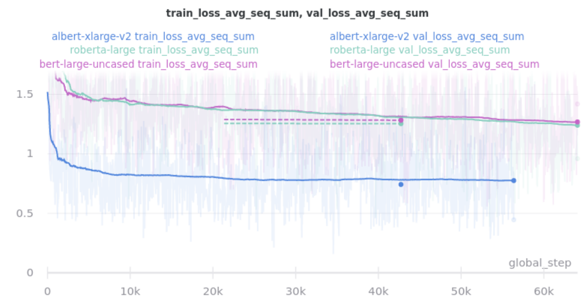
Outliers Included:

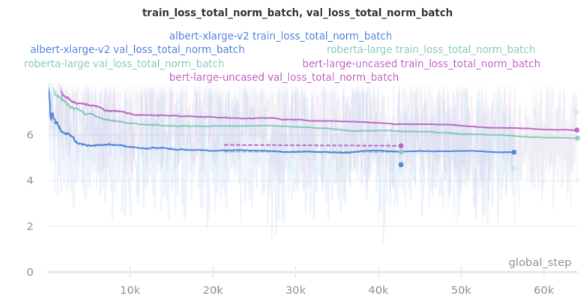


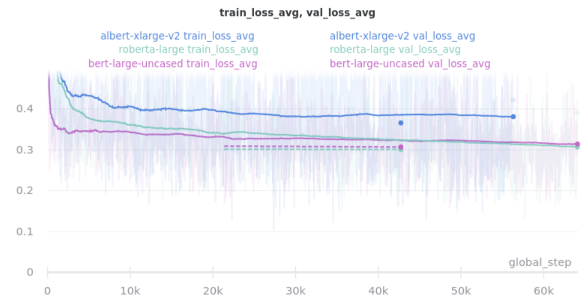

train_loss_total, val_loss_total
albert-xlarge-v2 train_loss_total
roberta-large train_loss_total
bert-large-uncased train_loss_total
albert-xlarge-v2 val_loss_total
roberta-large val_loss_total
bert-large-uncased val_loss_total
global_step

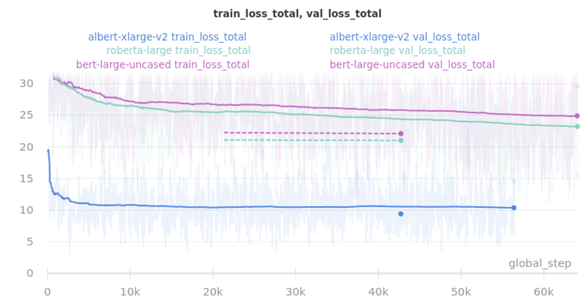
No Outliers:

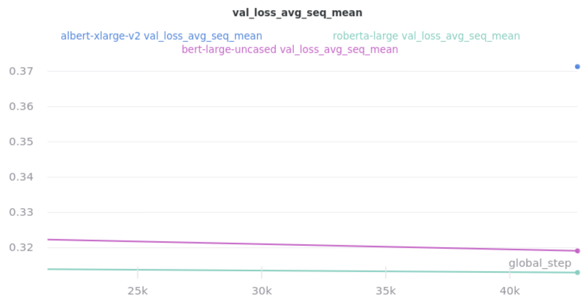

train_loss_avg_seq_mean, val_loss_avg_seq_mean
albert-xlarge-v2 train_loss_avg_seq_mean
roberta-large train_loss_avg_seq_mean
bert-large-uncased train_loss_avg_seq_mean
albert-xlarge-v2 val_loss_avg_seq_mean
roberta-large val_loss_avg_seq_mean
bert-large-uncased val_loss_avg_seq_mean
global_step


train_loss_avg_seq_sum, val_loss_avg_seq_sum
albert-xlarge-v2 train_loss_avg_seq_sum
roberta-large train_loss_avg_seq_sum
bert-large-uncased train_loss_avg_seq_sum
albert-xlarge-v2 val_loss_avg_seq_sum
roberta-large val_loss_avg_seq_sum
bert-large-uncased val_loss_avg_seq_sum
global_step


train_loss_total_norm_batch, val_loss_total_norm_batch
albert-xlarge-v2 train_loss_total_norm_batch
albert-xlarge-v2 val_loss_total_norm_batch
roberta-large train_loss_total_norm_batch
roberta-large val_loss_total_norm_batch
bert-large-uncased train_loss_total_norm_batch
bert-large-uncased val_loss_total_norm_batch
global_step

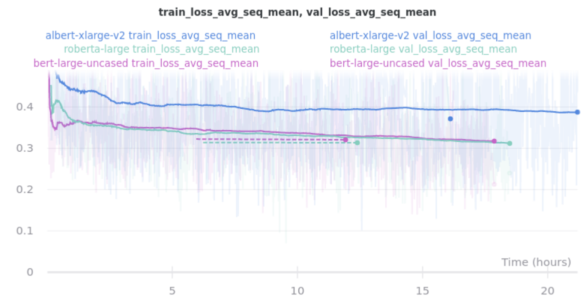

train_loss_avg, val_loss_avg
albert-xlarge-v2 train_loss_avg
roberta-large train_loss_avg
bert-large-uncased train_loss_avg
albert-xlarge-v2 val_loss_avg
roberta-large val_loss_avg
bert-large-uncased val_loss_avg
global_step





Relative Time:

This is included because the batch size for `albert-large-v2` had to be lowered to 2 (from 4).



11.3.3 Pooling Mode

See the main README.md for more information on what the pooling model is.

The two options, `sent_rep_tokens` and `mean_tokens`, were both tested with the `bert-base-uncased` and `distilbert-base-uncased` word embedding models.

Full command used to run the tests:

```
python main.py \
--model_name_or_path [Model Name] \
--model_type [Model Type] \
--pooling_mode [`mean_tokens` or `sent_rep_tokens`] \
--data_path ./cnn_dm_pt/[Model Type]-base \
--max_epochs 3 \
--accumulate_grad_batches 2 \
--warmup_steps 1800 \
--overfit_batches 0.6 \
--gradient_clip_val 1.0 \
--optimizer_type adamw \
```

(continues on next page)

(continued from previous page)

```
--use_scheduler linear \
--profiler \
--do_train --do_test \
--batch_size 16
```

Pooling Mode Results

Training Times and Model Sizes:

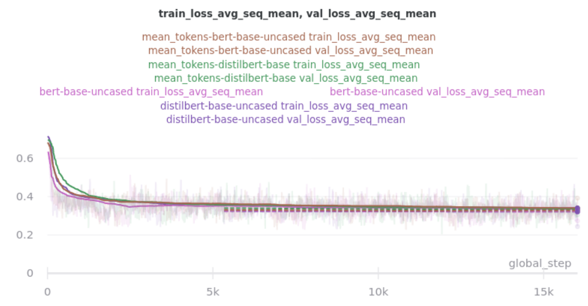
Model Key	Time	Model Size
`distilbert-base-uncased` mean_tokens	5h 18m 1s	810.6MB
`distilbert-base-uncased` sent_rep_tokens	4h 5m 30s	810.6MB
`bert-base-uncased` mean_tokens	8h 22m 46s	1.3GB
`bert-base-uncased` sent_rep_tokens	7h 56m 39s	1.3GB

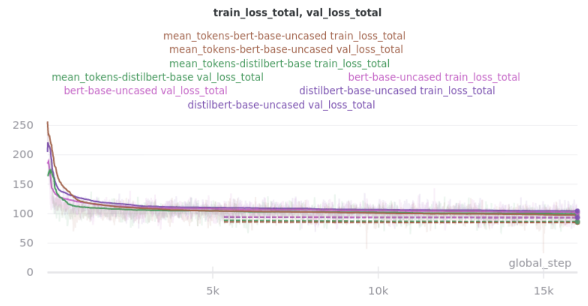
ROUGE Scores:

Name	ROUGE-1	ROUGE-2	ROUGE-L
distilbert-base-uncased mean_tokens	41.1	18.8	26.5
distilbert-base-uncased sent_rep_tokens	40.1	18.1	26.0
bert-base-uncased mean_tokens	40.7	18.7	26.6
bert-base-uncased sent_rep_tokens	40.2	18.2	26.1

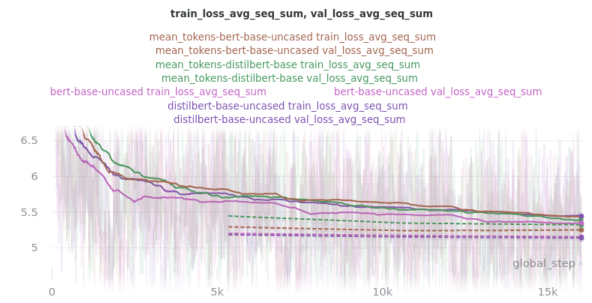
Main Takeaway: Using the `mean_tokens` `pooling_mode` is associated with a *0.617 average ROUGE F1 score improvement* over the `sent_rep_tokens` `pooling_mode`. This improvement is at the cost of a *49.3 average minute (2959 seconds) increase in training time*.

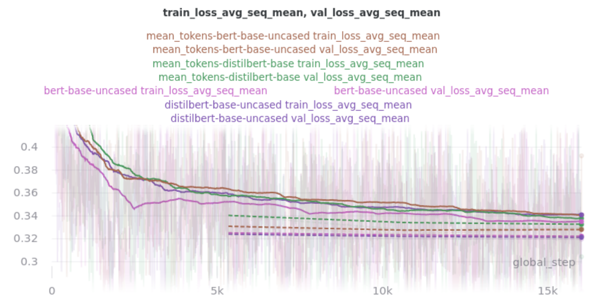
Outliers Included:

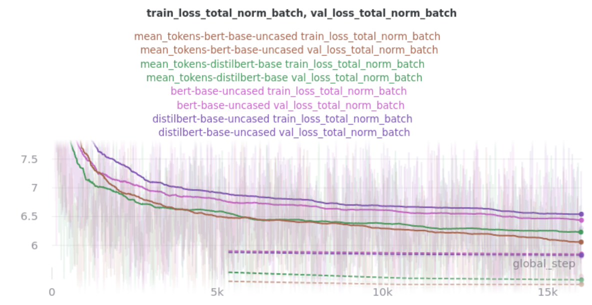


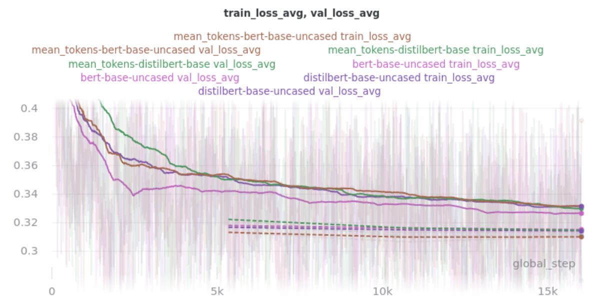


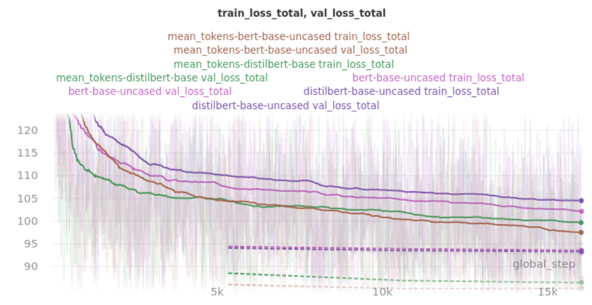
No Outliers:

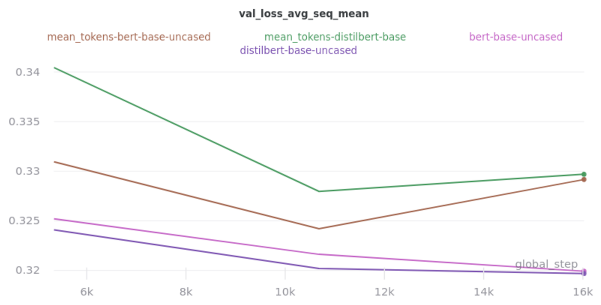

train_loss_avg_seq_sum, val_loss_avg_seq_sum
mean_tokens-bert-base-uncased train_loss_avg_seq_sum
mean_tokens-bert-base-uncased val_loss_avg_seq_sum
mean_tokens-distilbert-base train_loss_avg_seq_sum
mean_tokens-distilbert-base val_loss_avg_seq_sum
bert-base-uncased train_loss_avg_seq_sum
bert-base-uncased val_loss_avg_seq_sum
distilbert-base-uncased train_loss_avg_seq_sum
distilbert-base-uncased val_loss_avg_seq_sum
6.5
6
5.5
5
global_step
0
5k
10k
15k


train_loss_avg_seq_mean, val_loss_avg_seq_mean
mean_tokens-bert-base-uncased train_loss_avg_seq_mean
mean_tokens-bert-base-uncased val_loss_avg_seq_mean
mean_tokens-distilbert-base train_loss_avg_seq_mean
mean_tokens-distilbert-base val_loss_avg_seq_mean
bert-base-uncased train_loss_avg_seq_mean
bert-base-uncased val_loss_avg_seq_mean
distilbert-base-uncased train_loss_avg_seq_mean
distilbert-base-uncased val_loss_avg_seq_mean
0.4
0.38
0.36
0.34
0.32
0.3
global_step
0
5k
10k
15k

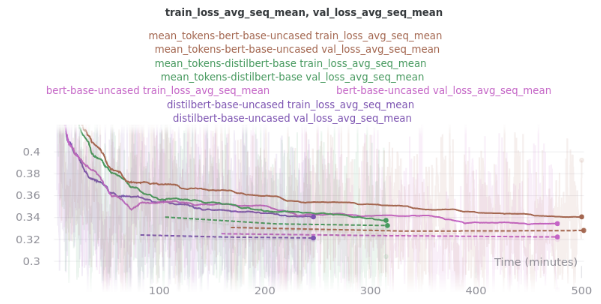

train_loss_total_norm_batch, val_loss_total_norm_batch
mean_tokens-bert-base-uncased train_loss_total_norm_batch
mean_tokens-bert-base-uncased val_loss_total_norm_batch
mean_tokens-distilbert-base train_loss_total_norm_batch
mean_tokens-distilbert-base val_loss_total_norm_batch
bert-base-uncased train_loss_total_norm_batch
bert-base-uncased val_loss_total_norm_batch
distilbert-base-uncased train_loss_total_norm_batch
distilbert-base-uncased val_loss_total_norm_batch
7.5
7
6.5
6
global_step
0
5k
10k
15k


train_loss_avg, val_loss_avg
mean_tokens-bert-base-uncased train_loss_avg
mean_tokens-bert-base-uncased val_loss_avg
mean_tokens-distilbert-base train_loss_avg
mean_tokens-distilbert-base val_loss_avg
bert-base-uncased train_loss_avg
bert-base-uncased val_loss_avg
distilbert-base-uncased train_loss_avg
distilbert-base-uncased val_loss_avg
0.4
0.38
0.36
0.34
0.32
0.3
global_step
0
5k
10k
15k


train_loss_total, val_loss_total
mean_tokens-bert-base-uncased train_loss_total
mean_tokens-bert-base-uncased val_loss_total
mean_tokens-distilbert-base train_loss_total
mean_tokens-distilbert-base val_loss_total
bert-base-uncased train_loss_total
bert-base-uncased val_loss_total
distilbert-base-uncased train_loss_total
distilbert-base-uncased val_loss_total
120
115
110
105
100
95
90
global_step
5k
10k
15k



Relative Time:



11.3.4 Classifier/Encoder

The classifier/encoder is responsible for removing the hidden features from each sentence embedding and converting them to a single number. The `linear`, `transformer` (with 2 layers), `transformer` (with 6 layers "`--classifier_transformer_num_layers 6`"), and `transformer_linear` options were tested with the `distilbert-base-uncased` model. The `transformer_linear` test has a transformer with *2 layers* (like the `transformer` test).

Unlike the experiments prior to this one (above), the "Classifier/Encoder" experiment used a `--train_percent_check` of 0.6, `--val_percent_check` of 0.6 and `--test_percent_check` of **1.0**. All of the data was used for testing whereas 60% of it was used for training and validation.

Full command used to run the tests:

```
python main.py \
--model_name_or_path [Model Name] \
--model_type distilbert \
--no_use_token_type_ids \
--classifier [`linear` or `transformer` or `transformer_linear`] \
[--classifier_transformer_num_layers 6 \]
--data_path ./cnn_dm_pt/bert-base-uncased \
--max_epochs 3 \
--accumulate_grad_batches 2 \
--warmup_steps 1800 \
--train_percent_check 0.6 --val_percent_check 0.6 --test_percent_check 1.0 \
--gradient_clip_val 1.0 \
--optimizer_type adamw \
--use_scheduler linear \
--profiler \
--do_train --do_test \
--batch_size 16
```

Classifier/Encoder Results

Training Times and Model Sizes:

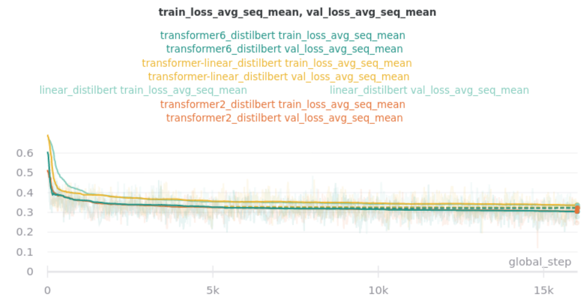
Model Key	Time	Model Size
`linear`	3h 59m 1s	810.6MB
`transformer` (2 layers)	4h 9m 29s	928.8MB
`transformer` (6 layers)	4h 21m 29s	1.2GB
`transformer_linear`	4h 9m 59s	943.0MB

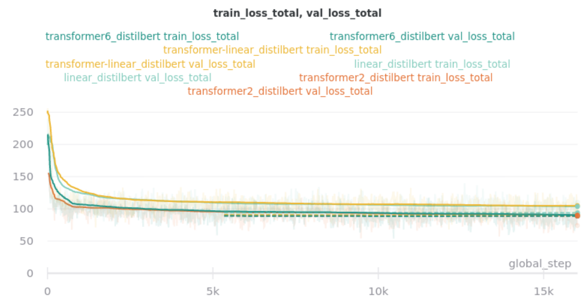
ROUGE Scores:

Name	ROUGE-1	ROUGE-2	ROUGE-L
`linear`	41.2	18.9	26.5
`transformer` (2 layers)	41.2	18.8	26.5
`transformer` (6 layers)	41.0	18.9	26.5
`transformer_linear`	40.9	18.7	26.6

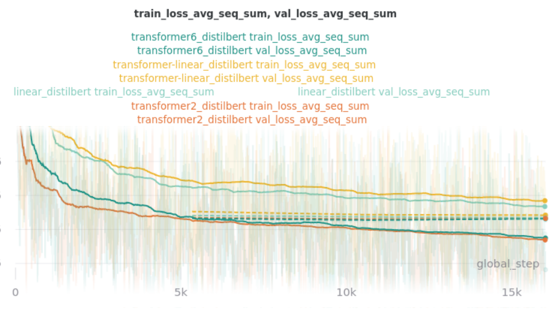
Main Takeaway: The `transformer` encoder had a much better loss curve, indicating that it is able to learn more about choosing the more representative sentences. However, its ROUGE scores are nearly identical to the `linear` encoder, which suggests both encoders capture enough information to summarize. The `transformer` encoder may potentially work better on more complex datasets.

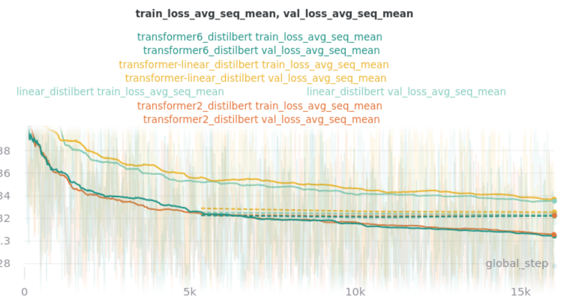
Outliers Included:

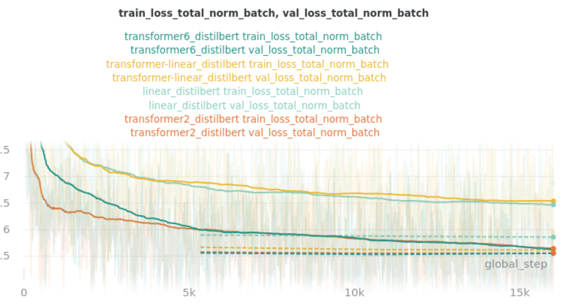


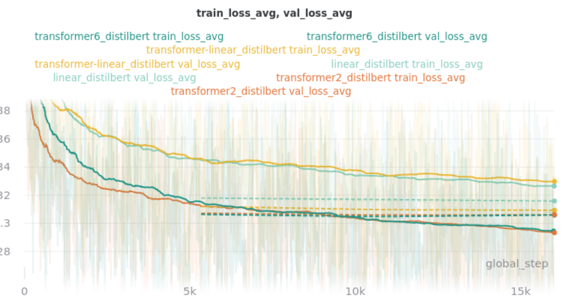


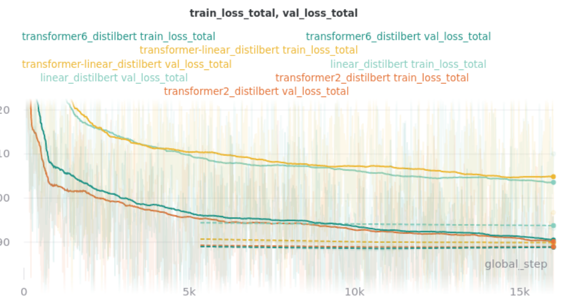
No Outliers:

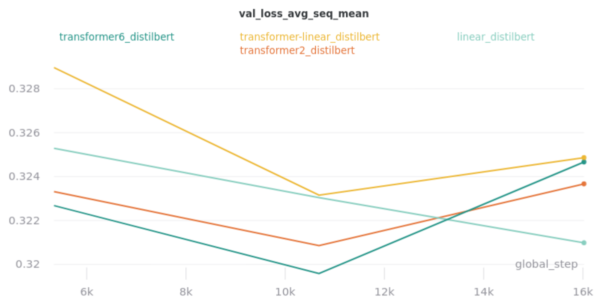

train_loss_avg_seq_sum, val_loss_avg_seq_sum
transformer6_distilbert train_loss_avg_seq_sum
transformer6_distilbert val_loss_avg_seq_sum
transformer-linear_distilbert train_loss_avg_seq_sum
transformer-linear_distilbert val_loss_avg_seq_sum
linear_distilbert train_loss_avg_seq_sum
linear_distilbert val_loss_avg_seq_sum
transformer2_distilbert train_loss_avg_seq_sum
transformer2_distilbert val_loss_avg_seq_sum
6
5.5
5
4.5
global_step
0
5k
10k
15k


train_loss_avg_seq_mean, val_loss_avg_seq_mean
transformer6_distilbert train_loss_avg_seq_mean
transformer6_distilbert val_loss_avg_seq_mean
transformer-linear_distilbert train_loss_avg_seq_mean
transformer-linear_distilbert val_loss_avg_seq_mean
linear_distilbert train_loss_avg_seq_mean
linear_distilbert val_loss_avg_seq_mean
transformer2_distilbert train_loss_avg_seq_mean
transformer2_distilbert val_loss_avg_seq_mean
0.38
0.36
0.34
0.32
0.3
0.28
global_step
0
5k
10k
15k

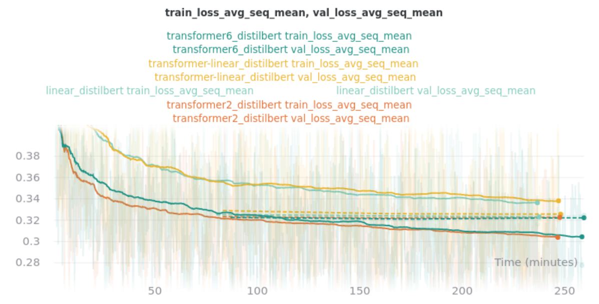

train_loss_total_norm_batch, val_loss_total_norm_batch
transformer6_distilbert train_loss_total_norm_batch
transformer6_distilbert val_loss_total_norm_batch
transformer-linear_distilbert train_loss_total_norm_batch
transformer-linear_distilbert val_loss_total_norm_batch
linear_distilbert train_loss_total_norm_batch
linear_distilbert val_loss_total_norm_batch
transformer2_distilbert train_loss_total_norm_batch
transformer2_distilbert val_loss_total_norm_batch
7.5
7
6.5
6
5.5
global_step
0
5k
10k
15k


train_loss_avg, val_loss_avg
transformer6_distilbert train_loss_avg
transformer6_distilbert val_loss_avg
transformer-linear_distilbert train_loss_avg
transformer-linear_distilbert val_loss_avg
linear_distilbert train_loss_avg
linear_distilbert val_loss_avg
transformer2_distilbert train_loss_avg
transformer2_distilbert val_loss_avg
0.38
0.36
0.34
0.32
0.3
0.28
global_step
0
5k
10k
15k


train_loss_total, val_loss_total
transformer6_distilbert train_loss_total
transformer6_distilbert val_loss_total
transformer-linear_distilbert train_loss_total
transformer-linear_distilbert val_loss_total
linear_distilbert train_loss_total
linear_distilbert val_loss_total
transformer2_distilbert train_loss_total
transformer2_distilbert val_loss_total
120
110
100
90
global_step
0
5k
10k
15k



Relative Time:



CHAPTER

TWELVE

EXTRACTIVE API REFERENCE

12.1 Model/Module

class extractive.**ExtractiveSummarizer**(*hparams*, *embedding_model_config=None*, *classifier_obj=None*)

A machine learning model that extractively summarizes an input text by scoring the sentences. Main class that handles the data loading, initial processing, training/testing/validating setup, and contains the actual model.

static add_model_specific_args(*parent_parser*)

Arguments specific to this model

compute_loss(*outputs*, *labels*, *mask*)

Compute the loss between model outputs and ground-truth labels.

Parameters

- **outputs** (*torch.Tensor*) – Output sentence scores obtained from *forward()*
- **labels** (*torch.Tensor*) – Ground-truth labels (1 for sentences that should be in the summary, `0` otherwise) from the batch generated during the data preprocessing stage.
- **mask** (*torch.Tensor*) – Mask returned by *forward()*, either `sent_rep_mask` or `sent_lengths_mask` depending on the pooling mode used during model initialization.

Returns

Losses: (total_loss, total_norm_batch_loss, sum_avg_seq_loss, mean_avg_seq_loss, average_loss)

Return type [tuple]

configure_optimizers()

Configure the optimizers. Returns the optimizer and scheduler specified by the values in `self.hparams`.

forward(*input_ids*, *attention_mask*, *sent_rep_mask=None*, *token_type_ids=None*, *sent_rep_token_ids=None*, *sent_lengths=None*, *sent_lengths_mask=None*, ***kwargs*)

Model forward function. See the 60 minute bliz tutorial if you are unsure what a forward function is.

Parameters

- **input_ids** (*torch.Tensor*) – Indices of input sequence tokens in the vocabulary. What are input IDs?
- **attention_mask** (*torch.Tensor*) – Mask to avoid performing attention on padding token indices. Mask values selected in `[0, 1]`: 1 for tokens that are NOT MASKED, `0` for MASKED tokens. What are attention masks?

- **sent_rep_mask** (*torch.Tensor, optional*) – Indicates which numbers in `sent_rep_token_ids` are actually the locations of sentence representation ids and which are padding. Defaults to None.
- **token_type_ids** (*torch.Tensor, optional*) – Usually, segment token indices to indicate first and second portions of the inputs. However, for summarization they are used to indicate different sentences. Depending on the size of the token type id vocabulary, these values may alternate between `0` and `1` or they may increase sequentially for each sentence in the input.. Defaults to None.
- **sent_rep_token_ids** (*torch.Tensor, optional*) – The locations of the sentence representation tokens. Defaults to None.
- **sent_lengths** (*torch.Tensor, optional*) – A list of the lengths of each sentence in `input_ids`. See `data.pad_batch_collate()` for more info about the generation of thisfeature. Defaults to None.
- **sent_lengths_mask** (*torch.Tensor, optional*) – Created on-the-fly by `data.pad_batch_collate()`. Similar to `sent_rep_mask`: `1` for value and `0` for padding. See `data.pad_batch_collate()` for more info about the generation of this feature. Defaults to None.

Returns Contains the sentence scores and mask as `torch.Tensors`. The mask is either the `sent_rep_mask` or `sent_lengths_mask` depending on the pooling mode used during model initialization.

Return type tuple

freeze_web_model()

Freezes the encoder `word_embedding_model`

json_to_dataset(*tokenizer, hparams, inputs=None, num_files=0, processor=None*)

Convert json output from `convert_to_extractive.py` to a ".pt" file containing lists or tensors using a `data.SentencesProcessor`. This function is run by `prepare_data()` in parallel.

Parameters

- **tokenizer** (*transformers.PreTrainedTokenizer*) – Tokenizer used to convert examples to input_ids. Usually is `self.tokenizer`.
- **hparams** (*argparse.Namespace*) – Hyper-parameters used to create the model. Usually is `self.hparams`.
- **inputs** (*tuple, optional*) – (idx, json_file) Current loop index and path to json file. Defaults to None.
- **num_files** (*int, optional*) – The total number of files to process. Used to display a nice progress indicator. Defaults to 0.
- **processor** (`data.SentencesProcessor`, *optional*) – The `data.SentencesProcessor` object to convert the json file to usable features. Defaults to None.

predict(*input_text: str, raw_scores=False, num_summary_sentences=3*)

Summarizes `input_text` using the model.

Parameters

- **input_text** (*str*) – The text to be summarized.
- **raw_scores** (*bool, optional*) – Return a list containing each sentence and its corespoding score instead of the summary. Defaults to False.

- `num_summary_sentences` (*int, optional*) – The number of sentences in the output summary. This value specifies the number of top sentences to select as the summary. Defaults to 3.

Returns The summary text. If `raw_scores` is set then returns a list of input sentences and their corespoding scores.

Return type str

predict_sentences(*input_sentences: Union[List[str], generator]*, *raw_scores=False*, *num_summary_sentences=3*, *tokenized=False*)

Summarizes `input_sentences` using the model.

Parameters

- `input_sentences` (*list or generator*) – The sentences to be summarized as a list or a generator of spacy Spans (`spacy.tokens.span.Span`), which can be obtained by running `nlp("input document").sents` where `nlp` is a spacy model with a sentencizer.
- `raw_scores` (*bool, optional*) – Return a list containing each sentence and its corespoding score instead of the summary. Defaults to False.
- `num_summary_sentences` (*int, optional*) – The number of sentences in the output summary. This value specifies the number of top sentences to select as the summary. Defaults to 3.
- `tokenized` (*bool, optional*) – If the input sentences are already tokenized using spacy. If true, `input_sentences` should be a list of lists where the outer list contains sentences and the inner lists contain tokens. Defaults to False.

Returns The summary text. If `raw_scores` is set then returns a list of input sentences and their corespoding scores.

Return type str

prepare_data()

Runs *json_to_dataset()* in parallel. *json_to_dataset()* is the function that actually loads and processes the examples as described below. Algorithm: For each json file outputted by the `convert_to_extractive.py` script:

1. Load json file.
2. **Add each document in json file to `SentencesProcessor` defined in `self.processor`,** overwriting any previous data in the processor.
3. **Run *data.SentencesProcessor.get_features()* to save the extracted features to disk** as a `.pt` file containing a pickled python list of dictionaries, which each dictionary contains the extracted features.

Memory Usage Note: If sharding was turned off during the `convert_to_extractive` process then this function will run once, loading the entire dataset into memory to process just like the `convert_to_extractive.py` script.

setup(*stage*)

Download the *word_embedding_model* if the model will be trained.

test_dataloader()

Create dataloader for testing.

test_epoch_end(*outputs*)

Called at the end of a testing epoch: PyTorch Lightning Documentation Finds the mean of all the metrics logged by *test_step()*.

test_step(*batch*, *batch_idx*)
Test step: PyTorch Lightning Documentation Similar to *validation_step()* in that in runs the inputs through the model. However, this method also calculates the ROUGE scores for each example-summary pair.

train_dataloader()
Create dataloader for training if it has not already been created.

training: bool

training_step(*batch*, *batch_idx*)
Training step: PyTorch Lightning Documentation

unfreeze_web_model()
Un-freezes the `word_embedding_model`

val_dataloader()
Create dataloader for validation.

validation_epoch_end(*outputs*)
Called at the end of a validation epoch: PyTorch Lightning Documentation Finds the mean of all the metrics logged by *validation_step()*.

validation_step(*batch*, *batch_idx*)
Validation step: PyTorch Lightning Documentation Similar to *training_step()* in that in runs the inputs through the model. However, this method also calculates accuracy and f1 score by marking every sentence score >0.5 as 1 (meaning should be in the summary) and each score <0.5 as 0 (meaning should not be in the summary).

extractive.**longformer_modifier**(*final_dictionary*)
Creates the `global_attention_mask` for the longformer. Tokens with global attention attend to all other tokens, and all other tokens attend to them. This is important for task-specific finetuning because it makes the model more flexible at representing the task. For example, for classification, the *<s>* token should be given global attention. For QA, all question tokens should also have global attention. For summarization, global attention is given to all of the *<s>* (RoBERTa 'CLS' equivalent) tokens. Please refer to the Longformer paper for more details. Mask values selected in `[0, 1]`: `0` for local attention, 1 for global attention.

12.2 Data

class data.**FSDataset**(*files_list*, *shuffle=True*, *verbose=False*)

get_files_lengths(*files_list*)

class data.**FSIterableDataset**(*files_list*, *shuffle=True*, *verbose=False*)
A dataset to yield examples from a list of files that are saved python objects that can be iterated over. These files could be other PyTorch datasets (tested with `TensorDataset`) or other python objects such as lists, for example. Each file will be loaded one at a time until all the examples have been yielded, at which point the next file will be loaded and used to yield examples, and so on. This means a large dataset can be broken into smaller chunks and this class can be used to load samples as if those files were one dataset while only utilizing the ram required for one chunk.

Explanation about `batch_size` and `__len__()`: If the `len()` function is needed to be accurate then the `batch_size` must be specified when constructing objects of this class. PyTorch `DataLoader` objects will report accurate lengths by dividing the number of examples in the dataset by the batch size only if the dataset if not an `IterableDataset`. If the dataset is an `IterableDataset` then a `DataLoader` will simply ask the dataset for its length, without diving by the batch size, because in some cases the length of an `IterableDataset` might be difficult or impossible to determine. However, in this case the number of examples (length of dataset) is known. The division by batch size must happen in the dataset (for datasets of type `IterableDataset`) since the `DataLoader` will not calculate this.

class `data.`**`InputExample`**(*text*, *labels*, *guid=None*, *target=None*)

`to_dict()`
Serializes this instance to a Python dictionary.

`to_json_string()`
Serializes this instance to a JSON string.

class `data.`**`InputFeatures`**(*input_ids*, *attention_mask=None*, *token_type_ids=None*, *labels=None*, *sent_rep_token_ids=None*, *sent_lengths=None*, *source=None*, *target=None*)
A single set of features of data.

Parameters

- **`input_ids`** – Indices of input sequence tokens in the vocabulary.
- **`attention_mask`** – Mask to avoid performing attention on padding token indices. Mask values selected in *[0, 1]*: Usually *1* for tokens that are NOT MASKED, *0* for MASKED (padded) tokens.
- **`token_type_ids`** – Usually, segment token indices to indicate first and second portions of the inputs. However, for summarization they are used to indicate different sentences. Depending on the size of the token type id vocabulary, these values may alternate between `0` and `1` or they may increase sequentially for each sentence in the input.
- **`labels`** – Labels corresponding to the input.
- **`sent_rep_token_ids`** – The locations of the sentence representation tokens.
- **`sent_lengths`** – A list of the lengths of each sentence in the *source* and *input_ids*.
- **`source`** – The actual source document as a list of sentences.
- **`target`** – The ground truth abstractive summary.

`to_dict()`
Serializes this instance to a Python dictionary.

`to_json_string()`
Serializes this instance to a JSON string.

class `data.`**`SentencesProcessor`**(*name=None*, *labels=None*, *examples=None*, *verbose=False*)
Create a *SentencesProcessor*

Parameters

- **`name`** (*`str, optional`*) – A label for the `SentencesProcessor` object, used internally for saving if a save name is not specified in *`data.SentencesProcessor.get_features()`*, Default is None.
- **`labels`** (*`list, optional`*) – The label that goes with each sample, can be a list of lists where the inside lists are the labels for each sentence in the coresponding example. Default is None.
- **`examples`** (*`list, optional`*) – List of `InputExamples`. Default is None.
- **`verbose`** (*`bool, optional`*) – Log extra information (such as examples of processed data points). Default is False.

`add_examples`(*texts*, *labels=None*, *ids=None*, *oracle_ids=None*, *targets=None*, *overwrite_labels=False*, *overwrite_examples=False*)

Primary method of adding example sets of texts, labels, ids, and targets to the `SentencesProcessor`

Parameters

- **`texts`** (*`list`*) – A list of documents where each document is a list of sentences where each sentence is a list of tokens. This is the output of *convert_to_extractive.py* and is in the 'src' field for each doc. See *`extractive.ExtractiveSummarizer.prepare_data()`*.
- **`labels`** (*`list, optional`*) – A list of the labels for each document where each label is a list of labels where the index of the label coresponds with the index of the sentence in the respective entry in *texts*. Similarly to *texts*, this is handled automatically by *ExtractiveSummarizer.prepare_data*. Default is None.
- **`ids`** (*`list, optional`*) – A list of ids for each document. Not used by *ExtractiveSummarizer*. Default is None.
- **`oracle_ids`** (*`list, optional`*) – Similar to labels but is a list of indexes of the chosen sentences instead of a one-hot encoded vector. These will be converted to labels. Default is None.
- **`targets`** (*`list, optional`*) – A list of the abstractive target for each document. Default is None.
- **`overwrite_labels`** (*`bool, optional`*) – Replace any labels currently stored by the `SentencesProcessor`. Default is False.
- **`overwrite_examples`** (*`bool, optional`*) – Replace any examples currently stored by the `SentencesProcessor`. Default is False.

Returns The examples as `InputExample`s that have been added.

Return type list

`classmethod create_from_examples`(*texts*, *labels=None*, ***kwargs*)

Create a SentencesProcessor with `**kwargs` and add `texts` and *labels*` through *`add_examples()`*.

`get_features`(*tokenizer*, *bert_compatible_cls=True*, *create_sent_rep_token_ids=True*, *sent_rep_token_id=None*, *create_sent_lengths=True*, *create_segment_ids='binary'*, *segment_token_id=None*, *create_source=False*, *n_process=2*, *max_length=None*, *pad_on_left=False*, *pad_token=0*, *mask_padding_with_zero=True*, *create_attention_mask=True*, *pad_ids_and_attention=True*, *return_type=None*, *save_to_path=None*, *save_to_name=None*, *save_as_type='txt'*)

Convert the examples stored by the `SentencesProcessor` to features that can be used by a model. The following processes can be performed: tokenization, token type ids (to separate sentences), sentence representation token ids (the locations of each sentence representation token), sentence lengths, and the attention mask. Padding can be applied to the tokenized examples and the attention masks but it is recommended to instead use the *`data.pad_batch_collate()`* function so each batch is padded individually for efficiency (less zeros passed through model).

Parameters

- **`tokenizer`** (*`transformers.PreTrainedTokenizer`*) – The tokenizer used to tokenize the examples.
- **`bert_compatible_cls`** (*`bool, optional`*) – Adds '[CLS]' tokens in front of each sentence. This is useful so that the '[CLS]' token can be used to obtain sentence embeddings. This only works if the chosen model has the '[CLS]' token in its vocabulary. Default is True.

- **create_sent_rep_token_ids** (*bool, optional*) – Option to create sentence representation token ids. This will store a list of the indexes of all the `sent_rep_token_ids` in the tokenized example. Default is True.
- **sent_rep_token_id** (*[type], optional*) – The token id that should be captured for each sentence (should have one per sentence and each should represent that sentence). Default is `'[CLS]' token if bert_compatible_cls else '[SEP]' token`.
- **create_sent_lengths** (*bool, optional*) – Option to create a list of sentence lengths where each index in the list coresponds to the respective sentence in the example. Default is True.
- **create_segment_ids** (*str, optional*) – Option to create segment ids (aka token type ids). See https://huggingface.co/transformers/glossary.html#token-type-ids for more info. Set to either "binary", "sequential", or False.
  - `binary` alternates between 0 and 1 for each sentence.
  - `sequential` starts at 0 and increments by 1 for each sentence.
  - `False` does not create any segment ids.

  Note: Many pretrained models that accept token type ids use them for question answering ans related tasks where the model receives two inputs. Therefore, most models have a token type id vocabulary size of 2, which means they only have learned 2 token type ids. The "binary" mode exists so that these pretrained models can easily be used. Default is "binary".
- **segment_token_id** (*str, optional*) – The token id to be used when creating segment ids. Can be set to 'period' to treat periods as sentence separation tokens, but this is a terrible idea for obvious reasons. Default is '[SEP]' token id.
- **create_source** (*bool, optional*) – Option to save the source text (non-tokenized) as a string. Default is False.
- **n_process** (*int, optional*) – How many processes to use for multithreading for running get_features_process(). Set higher to run faster and set lower is you experience OOM issues. Default is 2.
- **max_length** (*int, optional*) – If `pad_ids_and_attention` is True then pad to this amount. Default is `tokenizer.max_len`.
- **pad_on_left** (*bool, optional*) – Optionally, pad on the left instead of right. Default is False.
- **pad_token** (*int, optional*) – Which token to use for padding the `input_ids`. Default is 0.
- **mask_padding_with_zero** (*bool, optional*) – Use zeros to pad the attention. Uses ones otherwise. Default is True.
- **create_attention_mask** (*bool, optional*) – Option to create the attention mask. It is recommended to use the *`data.pad_batch_collate()`* function, which will automatically create attention masks and pad them on a per batch level. Default is `False if return_type == "lists" else True`.
- **pad_ids_and_attention** (*bool, optional*) – Pad the `input_ids` with `pad_token` and attention masks with 0s or 1s deneding on `mask_padding_with_zero`. Pad both to `max_length`. Default is `False if return_type == "lists" else True`
- **return_type** (*str, optional*) – Either "tensors", "lists", or None. See "Returns" section below. Default is None.

- **save_to_path** (*str, optional*) – The folder/directory to save the data to OR None to not save. Will save the data specified by `return_type` to disk. Default is None.
- **save_to_name** (*str, optional*) – The name of the file to save. The extension '.pt' is automatically appended. Default is `'dataset_' + self.name + '.pt'`.
- **save_as_type** (*str, optional*) – The file extension of saved file if *save_to_path* is set. Supports "pt" (PyTorch) and "txt" (Text). Saving as "txt" requires the `return_type` to be `lists`. If `return_type` is `tensors` the only `save_as_type` available is "pt". Defaults to "txt".

Returns If `return_type is None` return the list of calculated features. If `return_type == "tensors"` return the features converted to tensors and stacked such that features are grouped together into individual tensors. If `return_type == "lists"`, which is the recommended option then exports each `InputFeatures` object in the exported `features` list as a dictionary and appends each dictionary to a list. Returns that list.

Return type list or torch.TensorDataset

get_features_process(*input_information*, *num_examples=0*, *tokenizer=None*, *bert_compatible_cls=True*, *sep_token=None*, *cls_token=None*, *create_sent_rep_token_ids=True*, *sent_rep_token_id=None*, *create_sent_lengths=True*, *create_segment_ids='binary'*, *segment_token_id=None*, *create_source=False*, *max_length=None*, *pad_on_left=False*, *pad_token=0*, *mask_padding_with_zero=True*, *create_attention_mask=True*, *pad_ids_and_attention=True*)

The process that actually creates the features. `get_features()` is the driving function, look there for a description of how this function works. This function only exists so that processing can easily be done in parallel using `Pool.map`.

classmethod get_input_ids(*tokenizer*, *src_txt*, *bert_compatible_cls=True*, *sep_token=None*, *cls_token=None*, *max_length=None*)

Get `input_ids` from `src_txt` using `tokenizer`. See `get_features()` for more info.

load(*load_from_path*, *dataset_name=None*)

Attempts to load the dataset from storage. If that fails, will return None.

data.**pad_batch_collate**(*batch*, *modifier=None*)

Collate function to be passed to `DataLoaders`. PyTorch Docs: https://pytorch.org/docs/stable/data.html#dataloader-collate-fn

Calculates padding (per batch for efficiency) of `labels` and `token_type_ids` if they exist within the batch from the `Dataset`. Also, pads `sent_rep_token_ids` and creates the `sent_rep_mask` to indicate which numbers in the `sent_rep_token_ids` list are actually the locations of sentence representation ids and which are padding. Finally, calculates the `attention_mask` for each set of `input_ids` and pads both the `attention_mask` and the `input_ids`. Converts all inputs to tensors.

If `sent_lengths` are found then they will also automatically be padded. However, the padding for sentence lengths is complicated. Each list of sentence lengths needs to be the length of the longest list of sentence lengths and the sum of all the lengths in each list needs to add to the length of the input_ids width (the length of each input_id). The second requirement exists because `torch.split()` (which is used in the `mean_tokens` pooling algorithm to convert word vectors to sentence embeddings in `pooling.py`) will split a tensor into the lengths requested but will error instead of returning any extra. However, `torch.split()` will split a tensor into zero length segments. Thus, to solve this, zeros are added to each sentence length list for each example until one more padding value is needed to get the maximum number of sentences. Once only one more value is needed, the total value needded to reach the width of the `input_ids` is added.

`source` and `target`, if present, are simply passed on without any processing. Therefore, the standard `collate_fn` function for `DataLoaders` will not work if these are present since they cannot be converted to tensors without padding. This `collate_fn` must be used if `source` or `target` is present in the loaded dataset.

The `modifier` argument accepts a function that takes the `final_dictionary` and returns a modified `final_dictionary`. The `modifier` function will be called directly before `final_dictionary` is returned in *`pad_batch_collate()`*. This allows for easy extendability.

12.3 Pooling

`class pooling.`**`Pooling`**(*sent_rep_tokens=True*, *mean_tokens=False*, *max_tokens=False*)

Methods to obtains sentence embeddings from word vectors. Multiple methods can be specificed and their results will be concatenated together.

Parameters

- **`sent_rep_tokens`** (*`bool, optional`*) – Use the sentence representation token as sentence embeddings. Default is True.
- **`mean_tokens`** (*`bool, optional`*) – Take the mean of all the token vectors in
- **`False.`** (*`each sentence. Default is`*) –

`forward`(*word_vectors=None*, *sent_rep_token_ids=None*, *sent_rep_mask=None*, *sent_lengths=None*, *sent_lengths_mask=None*)

Forward pass of the Pooling nn.Module.

Parameters

- **`word_vectors`** (*`torch.Tensor, optional`*) – Vectors representing words created by a `word_embedding_model`. Defaults to None.
- **`sent_rep_token_ids`** (*`torch.Tensor, optional`*) – See *`extractive.ExtractiveSummarizer.forward()`*. Defaults to None.
- **`sent_rep_mask`** (*`torch.Tensor, optional`*) – See *`extractive.ExtractiveSummarizer.forward()`*. Defaults to None.
- **`sent_lengths`** (*`torch.Tensor, optional`*) – See *`extractive.ExtractiveSummarizer.forward()`*. Defaults to None.
- **`sent_lengths_mask`** (*`torch.Tensor, optional`*) – See *`extractive.ExtractiveSummarizer.forward()`*. Defaults to None.

Returns (output_vector, output_mask) Contains the sentence scores and mask as `torch.Tensors`. The mask is either the `sent_rep_mask` or `sent_lengths_mask` depending on the pooling mode used during model initialization.

Return type tuple

`training: bool`

12.4 Classifier

`class classifier.`**`LinearClassifier`**(*web_hidden_size*, *linear_hidden=1536*, *dropout=0.1*, *activation_string='gelu'*)

`nn.Module` to classify sentences by reducing the hidden dimension to 1.

Parameters

- **`web_hidden_size`** (*`int`*) – The output hidden size from the word embedding model. Used as the input to the first linear layer in this nn.Module.

- **linear_hidden** (*int, optional*) – The number of hidden parameters for this Classifier. Default is 1536.
- **dropout** (*float, optional*) – The value for dropout applied before the 2nd linear layer. Default is 0.1.
- **activation_string** (*str, optional*) – A string representing an activation function in `get_activation()` Default is "gelu".

forward(*x*, *mask*)
Forward function. `x` is the input `sent_vector` tensor and `mask` avoids computations on padded values. Returns `sent_scores`.

training: bool

class classifier.**SimpleLinearClassifier**(*web_hidden_size*)
`nn.Module` to classify sentences by reducing the hidden dimension to 1. This module contains a single linear layer and a sigmoid.

Parameters **web_hidden_size** (*int*) – The output hidden size from the word embedding model. Used as the input to the first linear layer in this nn.Module.

forward(*x*, *mask*)
Forward function. `x` is the input `sent_vector` tensor and `mask` avoids computations on padded values. Returns `sent_scores`.

training: bool

class classifier.**TransformerEncoderClassifier**(*d_model*, *nhead=8*, *dim_feedforward=2048*, *dropout=0.1*, *num_layers=2*, *custom_reduction=None*)
`nn.Module` to classify sentences by running the sentence vectors through some `nn.TransformerEncoder` layers and then reducing the hidden dimension to 1 with a linear layer.

Parameters

- **d_model** (*int*) – The number of expected features in the input
- **nhead** (*int, optional*) – The number of heads in the multiheadattention models. Default is 8.
- **dim_feedforward** (*int, optional*) – The dimension of the feedforward network model. Default is 2048.
- **dropout** (*float, optional*) – The dropout value. Default is 0.1.
- **num_layers** (*int, optional*) – The number of `TransformerEncoderLayers`. Default is 2.
- **reduction** (*nn.Module, optional*) – a nn.Module that maps *d_model* inputs to 1 value; if not specified then a `nn.Sequential()` module consisting of a linear layer and a sigmoid will automatically be created. Default is `nn.Sequential(linear, sigmoid)`.

forward(*x*, *mask*)
Forward function. `x` is the input `sent_vector` tensor and `mask` avoids computations on padded values. Returns `sent_scores`.

training: bool

12.5 Convert To Extractive

`convert_to_extractive.`**`cal_rouge`**(*evaluated_ngrams*, *reference_ngrams*)

`convert_to_extractive.`**`check_resume_success`**(*nlp*, *args*, *source_file*, *last_shard*, *output_path*, *split*, *compression*)

`convert_to_extractive.`**`combination_selection`**(*doc_sent_list*, *abstract_sent_list*, *summary_size*)

`convert_to_extractive.`**`convert_to_extractive_driver`**(*args*)

Driver function to convert an abstractive summarization dataset to an extractive dataset. The abstractive dataset must be formatted with two files for each split: a source and target file. Example file list for two splits: `["train.source", "train.target", "val.source", "val.target"]`

`convert_to_extractive.`**`convert_to_extractive_process`**(*args*, *nlp*, *source_docs*, *target_docs*, *name*, *piece_idx=None*)

Main process to convert an abstractive summarization dataset to extractive. Tokenizes, gets the `oracle_ids`, splits into `source` and `labels`, and saves processed data.

`convert_to_extractive.`**`example_processor`**(*inputs*, *args*, *oracle_mode='greedy'*, *no_preprocess=False*)

Create `oracle_ids`, convert them to `labels` and run `preprocess()`.

`convert_to_extractive.`**`greedy_selection`**(*doc_sent_list*, *abstract_sent_list*, *summary_size*)

`convert_to_extractive.`**`preprocess`**(*example*, *labels*, *min_sentence_ntokens=5*, *max_sentence_ntokens=200*, *min_example_nsents=3*, *max_example_nsents=100*)

Removes sentences that are too long/short and examples that have too few/many sentences.

`convert_to_extractive.`**`read_in_chunks`**(*file_object*, *chunk_size=5000*)

Read a file line by line but yield chunks of `chunk_size` number of lines at a time.

`convert_to_extractive.`**`resume`**(*output_path*, *split*, *chunk_size*)

Find the last shard created and return the total number of lines read and last shard number.

`convert_to_extractive.`**`save`**(*json_to_save*, *output_path*, *compression=False*)

Save `json_to_save` to `output_path` with optional gzip compresssion specified by `compression`.

`convert_to_extractive.`**`seek_files`**(*files*, *line_num*)

Seek a set of files to line number `line_num` and return the files.

`convert_to_extractive.`**`tokenize`**(*nlp*, *docs*, *n_process=5*, *batch_size=100*, *name=''*, *tokenizer_log_interval=0.1*, *disable_progress_bar=False*)

Tokenize using spacy and split into sentences and tokens.

CHAPTER

THIRTEEN

ABSTRACTIVE PRE-TRAINED MODELS & RESULTS

13.1 BART Converted to LongformerEncoderDecoder

Important: The models in this section are the output from the convert_bart_to_longformerencoderdecoder.py script without any gradient updates. This means that they need to be fine-tuned on a long document summarization dataset, such as Arxiv-PubMed, in order to create a model that can summarize long sequences.

The additional position embeddings for these models were initialized by copying the embeddings of the first `512` positions. This initialization is crucial for the model performance (check table 6 in the longformer paper for performance without this initialization).

The models output from the `convert_bart_to_longformerencoderdecoder.py` script do not work for long documents without further training. Tables 6 and 11 in the longformer paper suggest that models converted to be able to handle long content may perform well before any additional gradient updates. However, this does not appear to be true for summarization. The converted `facebook/bart-large-cnn` model from `huggingface/transformers` (aka `longformer-encdec-bart-large-cnn-converted`) produces almost random summaries that rarely pertain to the input document. Thus, these models need to be fine-tuned on a long document summarization dataset.

These are `huggingface/transformers` models, so they need to be used with the `--model_name_or_path` option. They can also be loaded directly in `huggingface/transformers` using `LEDForConditionalGeneration.from_pretrained()`.

The Google Drive folder containing my contributions to the below models is available at this link.

Name (Shortcut Code)	Initialized From	GDrive Download
allenai/led-base-16384	facebook/bart-base	
allenai/led-large-16384	facebook/bart-large	
HHousen/distil-led-large-cnn-16384	sshleifer/distilbart-cnn-12-6	Folder Link

Note: In pervious versions of TransformerSum, this section listed models that could be used with the outdated LED model (using custom versions of `huggingface/transformers` and `allenai/longformer`). Those models can still be found in this Google Drive Folder.

13.2 arXiv-PubMed

Name	Comments	Model Download	Data Download
led-base-4096-arxiv-pubmed	None	Model & All Checkpoints	Not yet..
led-large-4096-arxiv-pubmed	None	Not yet…	Not yet..
led-base-16384-arxiv-pubmed	None	Not yet…	Not yet..
led-large-16384-arxiv-pubmed	None	Not yet…	Not yet..

13.2.1 arXiv-PubMed ROUGE Scores

Test set results on the arXiv-PubMed dataset using ROUGE F_1.

Name	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-L-Sum
led-base-4096-arxiv-pubmed	Not yet…	Not yet…	Not yet…	Not yet…
led-large-4096-arxiv-pubmed	Not yet…	Not yet…	Not yet…	Not yet…
led-base-16384-arxiv-pubmed	Not yet…	Not yet…	Not yet…	Not yet…
led-large-16384-arxiv-pubmed	Not yet…	Not yet…	Not yet…	Not yet…

13.3 Individual ArXiv and PubMed models

The huggingface model hub has two pre-trained models for long text summarization: allenai/led-large-16384-arxiv and patrickvonplaten/led-large-16384-pubmed. These models can be used with pipelines to easily summarize long documents. Please see their model cards (by clicking on their names above) for more information.

CHAPTER

FOURTEEN

ABSTRACTIVE SUPPORTED DATASETS

All of the summarization datasets from the huggingface/nlp library are supported. Only 4 options (specifically `--dataset`, `--dataset_version`, `--data_example_column`, and `--data_summarized_column`) have to be changed to train a model on a new dataset.

The most notable datasets (the ones pertaining to summarization) are listed below.

> **Warning:** The `nlp` library uses arrow files which are not heavily compressed and can become large quite quickly. Thus, depending on your internet connection, hardware, and the size of the dataset it might be faster to reprocess the data than to download the pre-processed data.

If you download the preprocessed data, you can use it by setting the `--cache_file_path` option to the path containing the `train_tokenized`, `validation_tokenized`, and `test_tokenized` files.

Dataset Name	Processing Time	Preprocessed Data Download
`cnn_dailymail`	30m / 59m	bert-base-uncased & longformer-encdec-base-4096
`scientific_papers`	2h-4h	longformer-encdec-base-4096 & longformer-encdec-base-8192
`newsroom`		
`reddit`		
`multi_news`		
`gigaword`		
`billsum`		
`wikihow`		
`redit_tifu`		
`xsum`		
`opinosis`		

The live nlp viewer visualizes the data and describes each dataset.

14.1 Custom Datasets

You can use a custom dataset with the abstractive training script. The data needs to be stored in three Apache Arrow files: training, validation, and testing. You can specify the article and summary columns with `--data_example_column` and `--data_summarized_column`, respectfully. The `--dataset` argument allows multiple paths to be specified like so: `--dataset /path/to/train /path/to/valid /path/to/test`. **The files must be specified in train, validation, test order and each path must have a slash ("/") in it.** Inputs without a slash are interpreted as `huggingface/nlp` dataset names. Essentially, if your data is stored in three Arrow files (one for each split) and has at least 2 columns with the article and summary, then it is supported.

A convince script is provided at `scripts/convert_to_arrow.py` to convert JSON datasets to the Arrow format. It will convert a list of JSON files to Arrow files and then combine them into one file. Each JSON file is sequentially loaded into RAM so only one JSON file will be in RAM at a time. The final combined file is memory mapped so RAM usage will be close to zero during the combination stage. The maximum memory usage throughout the duration of the script will be the size of the largest input JSON file.

Output of `python scripts/convert_to_arrow.py --help`:

```
usage: convert_to_arrow.py [-h] [--file_paths FILE_PATHS [FILE_PATHS ...]] [--save_path␣
↪SAVE_PATH]
                            [--cache_path_prefix CACHE_PATH_PREFIX] [--no_combine]

optional arguments:
    -h, --help            show this help message and exit
    --file_paths FILE_PATHS [FILE_PATHS ...]
                            The paths to the JSON files to convert to arrow and combine.
    --save_path SAVE_PATH
                            The path to save the combined arrow file to. Defaults to './
↪data.arrow'.
    --cache_path_prefix CACHE_PATH_PREFIX
                            The cache path and file name prefix for the converted JSON␣
↪files. Defaults to './data_chunk'.
    --no_combine          Don't combine the converted JSON files.
```

CHAPTER

FIFTEEN

TRAINING AN ABSTRACTIVE SUMMARIZATION MODEL

You can finetune/train abstractive summarization models such as BART and T5 with this script. You can also train models consisting of any encoder and decoder combination with an EncoderDecoderModel by specifying the `--decoder_model_name_or_path` option (the `--model_name_or_path` argument specifies the encoder when using this configuration).

Alternatives:

- While you can use this script to load a pre-trained BART or T5 model and perform inference, it is recommended to use a huggingface/transformers summarization pipeline.
- To summarize PDF documents efficiently check out HHousen/DocSum.
- To summarize documents and strings of text using PreSumm please visit HHousen/DocSum.
- You can also use the summarization examples in huggingface/transformers, which are similar to this script, to train a model for seq2seq tasks. Most notably, the huggingface scripts don't integrate with `nlp` for easy and efficient dataset processing or make it easy to train EncoderDecoderModels.

Note: Version 3.1.0 of huggingface/transformers enhances the encoder-decoder framework to allow for more encoder decoder model combinations such as Bert2GPT2, Roberta2Roberta, and Longformer2Roberta. Patrick von Platen has trained and tested some of these model combinations using custom scripts. His results can be found at this huggingface/models page.

The effectiveness of initializing sequence-to-sequence models with pre-trained checkpoints for sequence generation tasks was shown in Leveraging Pre-trained Checkpoints for Sequence Generation Tasks by Sascha Rothe, Shashi Narayan, Aliaksei Severyn. Results for EncoderDecoderModels can be found in this paper. This script should be able to produce similar results to this paper.

Important: This script acts like a data preparation, training loop logic, and evaluation wrapper around models from `huggingface/transformers`. For this reason, you can specify the `--save_hg_transformer` option, which will save the `huggingface/transformers` model whenever a checkpoint is saved using `model.save_pretrained(save_path)`. Then, the trained model can be loaded without the `TransformerSum` library using just `huggingface/transformers` in the future by running `AutoModelForSeq2SeqLM.from_pretrained()` (or `EncoderDecoderModel.from_pretrained()` if `--decoder_model_name_or_path` was used during training).

15.1 Example

Example training command:

```
python main.py \
--mode abstractive \
--model_name_or_path bert-base-uncased \
--decoder_model_name_or_path bert-base-uncased \
--cache_file_path data \
--max_epochs 4 \
--do_train --do_test \
--batch_size 4 \
--weights_save_path model_weights \
--no_wandb_logger_log_model \
--accumulate_grad_batches 5 \
--use_scheduler linear \
--warmup_steps 8000 \
--gradient_clip_val 1.0 \
--custom_checkpoint_every_n 300
```

This command will train and test a bert-to-bert model for abstractive summarization for 4 epochs with a batch size of 4. The weights are saved to `model_weights/` and will not be uploaded to wandb.ai due to the `--no_wandb_logger_log_model` option. The CNN/DM dataset (which is the default dataset) will be downloaded (and automatically processed) to `data/`. The gradients will be accumulated every 5 batches and training will be optimized by AdamW with a scheduler that warms up linearly for 8000 then decays. A checkpoint file will be saved every 300 steps.

Importantly, you can specify the `--tie_encoder_decoder` option to tie the weights of the encoder and decoder when using an `EncoderDecoderModel` architecture. Specifying this option is equivalent to the "share" architecture tested in Leveraging Pre-trained Checkpoints for Sequence Generation Tasks.

15.2 Abstractive Long Summarization

This script can perform abstractive summarization on long sequences using the LongformerEncoderDecoder model. `LongformerEncoderDecoder` is BART (paper) but with components from the longformer (paper) that enable it to operate with long sequences.

During the development phase of the LED, LED installation and usage was complicated. Now, it is as simple as setting the `--model_name_or_path` option to a model from the LED community models page.

Important: You can adjust the sequence length that the trained LED model will be able to handle by modifying the `--model_max_length` argument. This option controls the length of sequences during the data processing stage. So, this option will have no effect with pre-compiled datasets listed under the "Preprocessed Data Download" heading on *Abstractive Supported Datasets*.

GitHub issues that discussed the creation of `LongformerEncoderDecoder`:

1. huggingface/transformers #4406
2. allenai/longformer #28

15.2.1 Step-by-Step Instructions

1. Download dataset (2.8GB): `gdown https://drive.google.com/uc?id=1rRzLwCl-s84Ji4ZfvfKV_iWunOy_cbE2`
2. Extract (90GB): `tar -xzvf longformer-encdec-base-8192.tar.gz`
3. Training command:

```
python main.py \
--mode abstractive \
--model_name_or_path allenai/led-base-16384 \
--max_epochs 4 \
--dataset scientific_papers \
--do_train \
--precision 16 \
--amp_level O2 \
--sortish_sampler \
--batch_size 8 \
--gradient_checkpointing \
--label_smoothing 0.1 \
--accumulate_grad_batches 2 \
--use_scheduler linear \
--warmup_steps 16000 \
--gradient_clip_val 1.0 \
--cache_file_path longformer-encdec-base-8192 \
--nlp_cache_dir nlp-cache \
--custom_checkpoint_every_n 18000
```

4. The `--max_epochs`, `--batch_size`, `--accumulate_grad_batches`, `--warmup_steps`, and `--custom_checkpoint_every_n` values will need to be tweaked.

15.3 Script Help

Output of `python main.py --mode abstractive --help` (*generic options* removed):

```
usage: main.py [-h]
            [--model_name_or_path MODEL_NAME_OR_PATH]
            [--decoder_model_name_or_path DECODER_MODEL_NAME_OR_PATH]
            [--batch_size BATCH_SIZE] [--val_batch_size VAL_BATCH_SIZE]
            [--test_batch_size TEST_BATCH_SIZE]
            [--dataloader_num_workers DATALOADER_NUM_WORKERS] [--only_preprocess]
            [--no_prepare_data] [--dataset DATASET [DATASET ...]]
            [--dataset_version DATASET_VERSION] [--data_example_column DATA_EXAMPLE_
↪COLUMN]
            [--data_summarized_column DATA_SUMMARIZED_COLUMN]
            [--cache_file_path CACHE_FILE_PATH] [--split_char SPLIT_CHAR]
            [--use_percentage_of_data USE_PERCENTAGE_OF_DATA]
            [--save_percentage SAVE_PERCENTAGE] [--save_hg_transformer] [--test_use_
↪pyrouge]
            [--sentencizer] [--gen_max_len GEN_MAX_LEN] [--label_smoothing LABEL_
↪SMOOTHING]
            [--sortish_sampler] [--nlp_cache_dir NLP_CACHE_DIR] [--tie_encoder_decoder]
```

(continues on next page)

(continued from previous page)

```
optional arguments:
--model_name_or_path MODEL_NAME_OR_PATH
                        Path to pre-trained model or shortcut name. A list of shortcut␣
↪names
                        can be found at
                        https://huggingface.co/transformers/pretrained_models.html.␣
↪Community-
                        uploaded models are located at https://huggingface.co/models.␣
↪Default
                        is 'bert-base-uncased'.
--decoder_model_name_or_path DECODER_MODEL_NAME_OR_PATH
                        Path to pre-trained model or shortcut name to use as the decoder␣
↪if an
                        EncoderDecoderModel architecture is desired. If this option is␣
↪not
                        specified, the shortcut name specified by `--model_name_or_path`␣
↪is
                        loaded using the Seq2seq AutoModel. Default is 'bert-base-uncased
↪'.
--batch_size BATCH_SIZE
                        Batch size per GPU/CPU for training/evaluation/testing.
--val_batch_size VAL_BATCH_SIZE
                        Batch size per GPU/CPU for evaluation. This option overwrites
                        `--batch_size` for evaluation only.
--test_batch_size TEST_BATCH_SIZE
                        Batch size per GPU/CPU for testing. This option overwrites
                        `--batch_size` for testing only.
--dataloader_num_workers DATALOADER_NUM_WORKERS
                        The number of workers to use when loading data. A general place␣
↪to
                        start is to set num_workers equal to the number of CPUs on your
                        machine. More details here: https://pytorch-
                        lightning.readthedocs.io/en/latest/performance.html#num-workers
--only_preprocess     Only preprocess and write the data to disk. Don't train model.
--no_prepare_data     Don't download, tokenize, or prepare data. Only load it from files.
--dataset DATASET [DATASET ...]
                        The dataset name from the `nlp` library or a list of paths to␣
↪Apache
                        Arrow files (that can be loaded with `nlp`) in the order train,
                        validation, test to use for training/evaluation/testing. Paths␣
↪must
                        contain a '/' to be interpreted correctly. Default is `cnn_
↪dailymail`.
--dataset_version DATASET_VERSION
                        The version of the dataset specified by `--dataset`.
--data_example_column DATA_EXAMPLE_COLUMN
                        The column of the `nlp` dataset that contains the text to be
                        summarized. Default value is for the `cnn_dailymail` dataset.
--data_summarized_column DATA_SUMMARIZED_COLUMN
                        The column of the `nlp` dataset that contains the summarized␣
↪text.
                        Default value is for the `cnn_dailymail` dataset.
```

(continues on next page)

(continued from previous page)

```
--cache_file_path CACHE_FILE_PATH
                        Path to cache the tokenized dataset.
--split_char SPLIT_CHAR
                        If the `--data_summarized_column` is already split into␣
↪sentences then
                        use this option to specify which token marks sentence boundaries.
↪ If
                        the summaries are not split into sentences then spacy will be␣
↪used to
                        split them. The default is None, which means to use spacy.
--use_percentage_of_data USE_PERCENTAGE_OF_DATA
                        When filtering the dataset, only save a percentage of the data.␣
↪This is
                        useful for debugging when you don't want to process the entire␣
↪dataset.
--save_percentage SAVE_PERCENTAGE
                        Percentage (divided by batch_size) between 0 and 1 of the␣
↪predicted and
                        target summaries from the test set to save to disk during␣
↪testing. This
                        depends on batch size: one item from each batch is saved
                        `--save_percentage` percent of the time. Thus, you can expect
                        `len(dataset)*save_percentage/batch_size` summaries to be saved.
--save_hg_transformer
                        Save the `huggingface/transformers` model whenever a checkpoint␣
↪is
                        saved.
--test_use_pyrouge    Use `pyrouge`, which is an interface to the official ROUGE␣
↪software,
                        instead of the pure-python implementation provided by `rouge-
↪score`.
                        You must have the real ROUGE package installed. More details␣
↪about
                        ROUGE 1.5.5 here:
                        https://github.com/andersjo/pyrouge/tree/master/tools/ROUGE-1.5.
↪5. It
                        is recommended to use this option for official scores. The␣
↪`ROUGE-L`
                        measurements from `pyrouge` are equivalent to the `rougeLsum`
                        measurements from the default `rouge-score` package.
--sentencizer         Use a spacy sentencizer instead of a statistical model for sentence
                        detection (much faster but less accurate) during data␣
↪preprocessing;
                        see https://spacy.io/api/sentencizer.
--gen_max_len GEN_MAX_LEN
                        Maximum sequence length during generation while testing and when␣
↪using
                        the `predict()` function.
--label_smoothing LABEL_SMOOTHING
                        `LabelSmoothingLoss` implementation from OpenNMT
                        (https://bit.ly/2ObgVPP) as stated in the original paper
                        https://arxiv.org/abs/1512.00567.
```

(continues on next page)

(continued from previous page)

```
--sortish_sampler     Reorganize the input_ids by length with a bit of randomness. This␣
↪can
                        help to avoid memory errors caused by large batches by forcing␣
↪large
                        batches to be processed first.
--nlp_cache_dir NLP_CACHE_DIR
                        Directory to cache datasets downloaded using `nlp`. Defaults to
                        '~/nlp'.
--tie_encoder_decoder
                        Tie the encoder and decoder weights. Only takes effect when␣
↪using an
                        EncoderDecoderModel architecture with the
                        `--decoder_model_name_or_path` option. Specifying this option is
                        equivalent to the 'share' architecture tested in 'Leveraging Pre-
                        trained Checkpoints for Sequence Generation Tasks'
                        (https://arxiv.org/abs/1907.12461).
```

CHAPTER

SIXTEEN

ABSTRACTIVE API REFERENCE

16.1 Model/Module

`class abstractive.AbstractiveSummarizer(`*hparams*`)`

A machine learning model that abstractively summarizes an input text using a seq2seq model. Main class that handles the data loading, initial processing, training/testing/validating setup, and contains the actual model.

`abs_collate_fn``(`*batch*, *modifier=None*`)`

`static` **`add_model_specific_args`**`(`*parent_parser*`)`

Arguments specific to this model

`calculate_loss``(`*prediction_scores*, *labels*`)`

`configure_optimizers``()`

Configure the optimizers. Returns the optimizer and scheduler specified by the values in `self.hparams`.

`forward``(`*source=None*, *target=None*, *source_mask=None*, *target_mask=None*, *labels=None*, ***kwargs*`)`

Model forward function. See the 60 minute bliz tutorial if you are unsure what a forward function is.

Parameters

- **`source`** (`torch.LongTensor` of shape `(batch_size, sequence_length)`, optional) – Indices of input sequence tokens in the vocabulary for the encoder. What are input IDs? Defaults to None.
- **`target`** (`torch.LongTensor` of shape `(batch_size, target_sequence_length)`, optional) – Provide for sequence to sequence training to the decoder. Defaults to None.
- **`source_mask`** (`torch.FloatTensor` of shape `(batch_size, sequence_length)`, optional) – Mask to avoid performing attention on padding token indices for the encoder. Mask values selected in `[0, 1]`: `1` for tokens that are NOT MASKED, `0` for MASKED tokens. Defaults to None.
- **`target_mask`** (`torch.BoolTensor` of shape `(batch_size, tgt_seq_len)`, optional) – `source_mask` but for the target sequence. Is an attention mask. Defaults to None.
- **`labels`** (`torch.LongTensor` of shape `(batch_size, sequence_length)`, optional) – Labels for computing the masked language modeling loss for the decoder. Indices should be in `[-100, 0, ..., config.vocab_size]`. Tokens with indices set to `-100` are ignored (masked), the loss is only computed for the tokens with labels in `[0, ..., config.vocab_size]` Defaults to None.

Returns (cross_entropy_loss, prediction_scores) The cross entropy loss and the prediction scores, which are the scores for each token in the vocabulary for each token in the output.

Return type tuple

`ids_to_clean_text`(*generated_ids*, *replace_sep_with_q=False*)

Convert IDs generated from `tokenizer.encode` to a string using `tokenizer.batch_decode` and also clean up spacing and special tokens.

Parameters

- **generated_ids** (*list*) – A list examples where each example is a list of IDs generated from `tokenizer.encode`.
- **replace_sep_with_q** (*bool, optional*) – Replace the `self.tokenizer.sep_token` with "<q>". Useful for determineing sentence boundaries and calculating ROUGE scores. Defaults to False.

Returns A list of examples where each example is a string or just one string if only one example was passed to this function.

Return type list or string

`on_save_checkpoint`(*checkpoint*)

Save the model in the `huggingface/transformers` format when a checkpoint is saved.

`predict`(*input_sequence*)

Summaries `input_sequence` using the model. Can summarize a list of sequences at once.

Parameters **input_sequence** (*str or list[str]*) – The text to be summarized.

Returns The summary text.

Return type str or list[str]

`prepare_data`()

Create the data using the `huggingface/nlp` library. This function handles downloading, preprocessing, tokenization, and feature extraction.

`setup`(*stage*)

Load the data created by `prepare_data()`. The downloading and loading is broken into two functions since prepare_data is only called from global_rank=0, and thus is not suitable for state (self.something) assignment.

`test_dataloader`()

Create dataloader for testing.

`test_epoch_end`(*outputs*)

Called at the end of a testing epoch: PyTorch Lightning Documentation Finds the mean of all the metrics logged by `test_step()`.

`test_step`(*batch*, *batch_idx*)

Test step: PyTorch Lightning Documentation Similar to `validation_step()` in that in runs the inputs through the model. However, this method also calculates the ROUGE scores for each example-summary pair.

`train_dataloader`()

Create dataloader for training.

`training: bool`

`training_step`(*batch*, *batch_idx*)

Training step: PyTorch Lightning Documentation

`val_dataloader`()

Create dataloader for validation.

`validation_step`(*batch*, *batch_idx*)

Validation step: PyTorch Lightning Documentation `abstractive.`**`longformer_modifier`**(*final_dictionary*, *tokenizer*, *attention_window*)

Creates the *global_attention_mask* for the longformer. Tokens with global attention attend to all other tokens, and all other tokens attend to them. This is important for task-specific finetuning because it makes the model more flexible at representing the task. For example, for classification, the <*s*> token should be given global attention. For QA, all question tokens should also have global attention. For summarization, global attention is given to all of the <*s*> (RoBERTa 'CLS' equivalent) tokens. Please refer to the Longformer paper for more details. Mask values selected in `[0, 1]`: `0` for local attention, `1` for global attention.

`abstractive.`**`trim_batch`**(*input_ids*, *pad_token_id*, *attention_mask=None*)

Remove columns that are populated exclusively by `pad_token_id`.

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX