**Pre-training Environment Setup Details**

Packages used:

* transformers[sentencepiece]
* datasets
* sacrebleu
* rouge\_score

<https://huggingface.co/docs/transformers/en/tokenizer_summary>

**SentencePiece:**

All tokenization algorithms described so far have the same problem: It is assumed that the input text uses spaces to separate words. However, not all languages use spaces to separate words. One possible solution is to use language specific pre-tokenizers, e.g. XLM uses a specific Chinese, Japanese, and Thai pre-tokenizer). To solve this problem more generally, SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing (Kudo et al., 2018) treats the input as a raw input stream, thus including the space in the set of characters to use. It then uses the BPE or unigram algorithm to construct the appropriate vocabulary.

<https://huggingface.co/transformers/v4.4.2/migration.html>

**AutoTokenizers and pipelines now use fast (rust) tokenizers by default:**

The python and rust tokenizers have roughly the same API, but the rust tokenizers have a more complete feature set. The rust tokenizers do not accept integers in the encoding methods.

**How to obtain the same behavior as v3.x in v4.x:**

The requirement on the SentencePiece dependency has been lifted from the setup.py file. This means that the tokenizers that depend on the SentencePiece library will not be available with a standard transformers installation. We need to do the following now.

*pip install transformers[sentencepiece]*

<https://pypi.org/project/sacrebleu/>

**sacreBLEU:**

SacreBLEU (Post, 2018) provides hassle-free computation of shareable, comparable, and reproducible BLEU scores.

<https://pypi.org/project/rouge-score/>

**Overview:**

This is a native python implementation of ROUGE, designed to replicate results from the original perl package. ROUGE was originally introduced in the paper:

Lin, Chin-Yew. ROUGE: a Package for Automatic Evaluation of Summaries. In Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004), Barcelona, Spain, July 25 - 26, 2004.

There are ROUGE implementations available for Python, however some are not native python due to their dependency on the perl script, and others provide differing results when compared with the original implementation. This makes it difficult to directly compare with known results.

This package is designed to replicate perl results. It implements:

* ROUGE-N (N-gram) scoring
* ROUGE-L (Longest Common Subsequence) scoring

**Two flavors of ROUGE-L:**

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

* sentence-level: Compute longest common subsequence (LCS) between two pieces of text. Newlines are ignored. This is called rougeL in this package.
* 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 this package.

<https://pypi.org/project/datasets/>

**Datasets is a lightweight library from Hugging Face that provides two main features:**

* one-line dataloaders for many public datasets: one-liners to download and pre-process any of the number of datasets major public datasets (image datasets, audio datasets, text datasets in 467 languages and dialects, etc.) provided on the [HuggingFace Datasets Hub](https://huggingface.co/datasets).
* efficient data pre-processing: simple, fast and reproducible data pre-processing for the public datasets as well as your own local datasets in CSV, JSON, text, PNG, JPEG, WAV, MP3, Parquet, etc. With simple commands like *processed\_dataset = dataset.map(process\_example)*, efficiently prepare the dataset for inspection and ML model evaluation and training.

**Training Parameter Details**

**1.0-model-training**

trainer\_args = TrainingArguments(

    output\_dir='pegasus-samsum',

    num\_train\_epochs=10,

    warmup\_steps=200,

    per\_device\_train\_batch\_size=1,

    per\_device\_eval\_batch\_size=1,

    weight\_decay=0.01,

    logging\_steps=10,

    evaluation\_strategy='steps',

    eval\_steps=50,

    save\_steps=1e6,

    gradient\_accumulation\_steps=16

)

**1.0-model-training**

trainer\_args = TrainingArguments(

    output\_dir='pegasus-samsum',

    num\_train\_epochs=4,

    warmup\_steps=200,

    per\_device\_train\_batch\_size=1,

    per\_device\_eval\_batch\_size=1,

    weight\_decay=0.01,

    logging\_steps=10,

    evaluation\_strategy='steps',

    eval\_steps=50,

    save\_steps=1e6,

    gradient\_accumulation\_steps=16

)

**Important Hugging Face Resources**

<https://huggingface.co/docs/transformers/en/model_doc/auto>

**Auto Classes:**

In many cases, the architecture you want to use can be guessed from the name or the path of the pretrained model you are supplying to the from\_pretrained() method. AutoClasses are here to do this job for you so that you automatically retrieve the relevant model given the name/path to the pretrained weights/config/vocabulary.

Instantiating one of AutoConfig, AutoModel, and AutoTokenizer will directly create a class of the relevant architecture. For instance

*model = AutoModel.from\_pretrained("google-bert/bert-base-cased")*

will create a model that is an instance of BertModel.

There is one class of AutoModel for each task, and for each backend (PyTorch, TensorFlow, or Flax).

**AutoModel**

<https://huggingface.co/docs/transformers/v4.41.3/en/model_doc/auto#transformers.AutoModel>

This is a generic model class that will be instantiated as one of the base model classes of the library when created with the from\_pretrained() class method or the from\_config() class method.

Note: Loading a model from its configuration file does not load the model weights. It only affects the model’s configuration. Use from\_pretrained() to load the model weights.

**AutoModelForSeq2SeqLM**

<https://huggingface.co/docs/transformers/en/model_doc/auto#transformers.AutoModelForSeq2SeqLM>

This is a generic model class that will be instantiated as one of the model classes of the library (with a sequence-to-sequence language modeling head) when created with the from\_pretrained() class method or the from\_config() class method.

Note: Loading a model from its configuration file does not load the model weights. It only affects the model’s configuration. Use from\_pretrained() to load the model weights.

<https://huggingface.co/docs/transformers/en/main_classes/data_collator>

**Data Collator**

Data collators are objects that will form a batch by using a list of dataset elements as input. These elements are of the same type as the elements of train\_dataset or eval\_dataset. To be able to build batches, data collators may apply some processing (like padding).

**DataCollatorForSeq2Seq**

<https://huggingface.co/docs/transformers/en/main_classes/data_collator#transformers.DataCollatorForSeq2Seq>

Data collator that will dynamically pad the inputs received, as well as the labels.

<https://huggingface.co/docs/transformers/en/main_classes/trainer>

The Trainer class provides an API for feature-complete training in PyTorch, and it supports distributed training on multiple GPUs/TPUs, mixed precision for NVIDIA GPUs, AMD GPUs, and torch.amp for PyTorch.

Trainer goes hand-in-hand with the TrainingArguments class, which offers a wide range of options to customize how a model is trained. Together, these two classes provide a complete training API.

**TrainingArguments**

<https://huggingface.co/docs/transformers/v4.41.3/en/main_classes/trainer#transformers.TrainingArguments>