Image Captioning (vision-encoder-text-decoder model) training example

The following example showcases how to finetune a vision-encoder-text-decoder model for image captioning using the JAX/Flax backend, leveraging 🤗 Transformers library's FlaxVisionEncoderDecoderModel.

JAX/Flax allows you to trace pure functions and compile them into efficient, fused accelerator code on both GPU and TPU. Models written in JAX/Flax are **immutable** and updated in a purely functional way which enables simple and efficient model parallelism.

`run_image_captioning_flax.py` is a lightweight example of how to download and preprocess a dataset from the 🤗 Datasets library or use your own files (jsonlines or csv), then fine-tune one of the architectures above on it.

For custom datasets in `jsonlines` format please see: https://huggingface.co/docs/datasets/loading_datasets.html#json-files and you also will find examples of these below.

Download COCO dataset (2017)

This example uses COCO dataset (2017) through a custom dataset script, which requires users to manually download the COCO dataset before training.

```
mkdir data
cd data
wget http://images.cocodataset.org/zips/train2017.zip
wget http://images.cocodataset.org/zips/val2017.zip
wget http://images.cocodataset.org/zips/test2017.zip
wget http://images.cocodataset.org/annotations/annotations_trainval2017.zip
wget http://images.cocodataset.org/annotations/image_info_test2017.zip
cd ..
```

Create a model from a vision encoder model and a text decoder model

Next, we create a FlaxVisionEncoderDecoderModel instance from a pre-trained vision encoder (ViT) and a pre-trained text decoder (GPT2):

```
python3 create_model_from_encoder_decoder_models.py \
    --output_dir model \
    --encoder_model_name_or_path google/vit-base-patch16-224-in21k \
    --decoder_model_name_or_path gpt2
```

Train the model

Finally, we can run the example script to train the model:

```
python3 run_image_captioning_flax.py \
    --output_dir ./image-captioning-training-results \
    --model_name_or_path model \
    --dataset_name ydshieh/coco_dataset_script \
    --dataset_config_name=2017 \
    --data_dir $PWD/data \
    --image_column image_path \
    --caption_column caption \
    --do_train --do_eval --predict_with_generate \
    --num_train_epochs 1 \
    --eval_steps 500 \
    --learning_rate 3e-5 --warmup_steps 0 \
    --per_device_train_batch_size 32 \
    --per_device_eval_batch_size 32 \
    --overwrite_output_dir \
    --max_target_length 32 \
    --num_beams 8 \
    --preprocessing_num_workers 16 \
    --logging_steps 10 \
    --block_size 16384 \
    --push_to_hub
```

This should finish in about 1h30 on Cloud TPU, with validation loss and ROUGE2 score of 2.0153 and 14.64 respectively after 1 epoch. Training statistics can be accessed on Models.