Language model training examples

The following example showcases how to train a language model from scratch using the JAX/Flax backend.

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.

Masked language modeling

In the following, we demonstrate how to train a bi-directional transformer model using masked language modeling objective as introduced in [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](). More specifically, we demonstrate how JAX/Flax can be leveraged to pre-train `roberta-base` in Norwegian on a single TPUv3-8 pod.

The example script uses the 🤗 Datasets library. You can easily customize them to your needs if you need extra processing on your datasets.

To setup all relevant files for training, let's create a directory.

```
mkdir ./norwegian-roberta-base
```

Train tokenizer

In the first step, we train a tokenizer to efficiently process the text input for the model. Similar to how it is shown in How to train a new language model from scratch using Transformers and Tokenizers, we use a `ByteLevelBPETokenizer`. The tokenizer is trained on the complete Norwegian dataset of OSCAR and consequently saved in the cloned model directory. This can take up to 10 minutes depending on your hardware ☕.

```
from datasets import load_dataset
from tokenizers import trainers, Tokenizer, normalizers, ByteLevelBPETokenizer

# load dataset
dataset = load_dataset("oscar", "unshuffled_deduplicated_no", split="train")

# Instantiate tokenizer
tokenizer = ByteLevelBPETokenizer()

def batch_iterator(batch_size=1000):
    for i in range(0, len(dataset), batch_size):
        yield dataset[i: i + batch_size]["text"]

# Customized training
tokenizer.train_from_iterator(batch_iterator(), vocab_size=50265, min_frequency=2,
special_tokens=[
    "<s>",
    "<pad>",
    "</s>",
    "<unk>",
    "<mask>",
])
```

```
# Save files to disk
tokenizer.save("./norwegian-roberta-base/tokenizer.json")
```

Create configuration

Next, we create the model's configuration file. This is as simple as loading and storing `**roberta-base**` in the local model folder:

```
from transformers import RobertaConfig

config = RobertaConfig.from_pretrained("roberta-base", vocab_size=50265)
config.save_pretrained("./norwegian-roberta-base")
```

Great, we have set up our model repository. During training, we will automatically push the training logs and model weights to the repo.

Train model

Next we can run the example script to pretrain the model:

```
python run_mlm_flax.py \
    --output_dir="./norwegian-roberta-base" \
    --model_type="roberta" \
    --config_name="./norwegian-roberta-base" \
    --tokenizer_name="./norwegian-roberta-base" \
    --dataset_name="oscar" \
    --dataset_config_name="unshuffled_deduplicated_no" \
    --max_seq_length="128" \
    --weight_decay="0.01" \
    --per_device_train_batch_size="128" \
    --per_device_eval_batch_size="128" \
    --learning_rate="3e-4" \
    --warmup_steps="1000" \
    --overwrite_output_dir \
    --num_train_epochs="18" \
    --adam_beta1="0.9" \
    --adam_beta2="0.98" \
    --logging_steps="500" \
    --save_steps="2500" \
    --eval_steps="2500" \
    --push_to_hub
```

Training should converge at a loss and accuracy of 1.78 and 0.64 respectively after 18 epochs on a single TPUv3-8. This should take less than 18 hours. Training statistics can be accessed on tfhub.dev.

For a step-by-step walkthrough of how to do masked language modeling in Flax, please have a look at this google colab.

Causal language modeling

In the following, we demonstrate how to train an auto-regressive causal transformer model in JAX/Flax. More specifically, we pretrain a randomely initialized `gpt2` model in Norwegian on a single TPUv3-8. to pre-train 124M `gpt2` in Norwegian on a single TPUv3-8 pod.

The example script uses the 🤗 Datasets library. You can easily customize them to your needs if you need extra processing on your datasets.

To setup all relevant files for training, let's create a directory.

```
mkdir ./norwegian-gpt2
```

Train tokenizer

In the first step, we train a tokenizer to efficiently process the text input for the model. Similar to how it is shown in [How to train a new language model from scratch using Transformers and Tokenizers](), we use a `ByteLevelBPETokenizer`. The tokenizer is trained on the complete Norwegian dataset of OSCAR and consequently saved in the cloned model directory. This can take up to 10 minutes depending on your hardware ☕.

```
from datasets import load_dataset
from tokenizers import trainers, Tokenizer, normalizers, ByteLevelBPETokenizer

# load dataset
dataset = load_dataset("oscar", "unshuffled_deduplicated_no", split="train")

# Instantiate tokenizer
tokenizer = ByteLevelBPETokenizer()

def batch_iterator(batch_size=1000):
    for i in range(0, len(dataset), batch_size):
        yield dataset[i: i + batch_size]["text"]

# Customized training
tokenizer.train_from_iterator(batch_iterator(), vocab_size=50257, min_frequency=2,
special_tokens=[
    "<s>",
    "<pad>",
    "</s>",
    "<unk>",
    "<mask>",
])

# Save files to disk
tokenizer.save("./norwegian-gpt2/tokenizer.json")
```

Create configuration

Next, we create the model's configuration file. This is as simple as loading and storing `**gpt2**` in the local model folder:

```
from transformers import GPT2Config
```

```
config = GPT2Config.from_pretrained("gpt2", resid_pdrop=0.0, embd_pdrop=0.0,
attn_pdrop=0.0, vocab_size=50257)
config.save_pretrained("./norwegian-gpt2")
```

Great, we have set up our model repository. During training, we will now automatically push the training logs and model weights to the repo.

Train model

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

```
python run_clm_flax.py \
    --output_dir="./norwegian-gpt2" \
    --model_type="gpt2" \
    --config_name="./norwegian-gpt2" \
    --tokenizer_name="./norwegian-gpt2" \
    --dataset_name="oscar" \
    --dataset_config_name="unshuffled_deduplicated_no" \
    --do_train --do_eval \
    --block_size="512" \
    --per_device_train_batch_size="64" \
    --per_device_eval_batch_size="64" \
    --learning_rate="5e-3" --warmup_steps="1000" \
    --adam_beta1="0.9" --adam_beta2="0.98" --weight_decay="0.01" \
    --overwrite_output_dir \
    --num_train_epochs="20" \
    --logging_steps="500" \
    --save_steps="2500" \
    --eval_steps="2500" \
    --push_to_hub
```

Training should converge at a loss and perplexity of 3.24 and 25.72 respectively after 20 epochs on a single TPUv3-8. This should take less than ~21 hours. Training statistics can be accessed on tfhub.de.

For a step-by-step walkthrough of how to do causal language modeling in Flax, please have a look at this google colab.

T5-like span-masked language modeling

In the following, we demonstrate how to train a T5 model using the span-masked language model objective as proposed in the Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. More specifically, we demonstrate how JAX/Flax can be leveraged to pre-train `google/t5-v1_1-base` in Norwegian on a single TPUv3-8 pod.

The example script uses the 🤗 Datasets library. You can easily customize them to your needs if you need extra processing on your datasets.

Let's start by creating a model repository to save the trained model and logs. Here we call the model `"norwegian-t5-base"`, but you can change the model name as you like.

To setup all relevant files for training, let's create a directory.

```
cd ./norwegian-t5-base
```

Train tokenizer

In the first step, we train a tokenizer to efficiently process the text input for the model. We make use of the [tokenizers](#) library to train a sentencepiece unigram tokenizer as shown in [t5_tokenizer_model.py](#) which is heavily inspired from [yandex-research/DeDLOC's tokenizer model](#) .

The tokenizer is trained on the complete Norwegian dataset of OSCAR and consequently saved in the cloned model directory. This can take up to 120 minutes depending on your hardware ☕☕☕ .

```
import datasets

from t5_tokenizer_model import SentencePieceUnigramTokenizer


vocab_size = 32_000
input_sentence_size = None

# Initialize a dataset
dataset = datasets.load_dataset("oscar", name="unshuffled_deduplicated_no",
split="train")

tokenizer = SentencePieceUnigramTokenizer(unk_token="<unk>", eos_token="</s>",
pad_token="<pad>")


# Build an iterator over this dataset
def batch_iterator(input_sentence_size=None):
    if input_sentence_size is None:
        input_sentence_size = len(dataset)
    batch_length = 100
    for i in range(0, input_sentence_size, batch_length):
        yield dataset[i: i + batch_length]["text"]


# Train tokenizer
tokenizer.train_from_iterator(
    iterator=batch_iterator(input_sentence_size=input_sentence_size),
    vocab_size=vocab_size,
    show_progress=True,
)

# Save files to disk
tokenizer.save("./norwegian-t5-base/tokenizer.json")
```

Create configuration

Next, we create the model's configuration file. This is as simple as loading and storing [`**google/t5-v1_1-base**`](#) in the local model folder:

```python
from transformers import T5Config

config = T5Config.from_pretrained("google/t5-v1_1-base", vocab_size=tokenizer.get_vocab_size())
config.save_pretrained("./norwegian-t5-base")
```

Great, we have set up our model repository. During training, we will automatically push the training logs and model weights to the repo.

Train model

Next we can run the example script to pretrain the model:

```bash
python run_t5_mlm_flax.py \
    --output_dir="./norwegian-t5-base" \
    --model_type="t5" \
    --config_name="./norwegian-t5-base" \
    --tokenizer_name="./norwegian-t5-base" \
    --dataset_name="oscar" \
    --dataset_config_name="unshuffled_deduplicated_no" \
    --max_seq_length="512" \
    --per_device_train_batch_size="32" \
    --per_device_eval_batch_size="32" \
    --adafactor \
    --learning_rate="0.005" \
    --weight_decay="0.001" \
    --warmup_steps="2000" \
    --overwrite_output_dir \
    --logging_steps="500" \
    --save_steps="10000" \
    --eval_steps="2500" \
    --push_to_hub
```

Training should converge at a loss and accuracy of 2.36 and 57.0 respectively after 3 epochs on a single TPUv3-8. This should take around 4.5 hours. Training statistics can be accessed on directly on the 🤗 hub

Runtime evaluation

We also ran masked language modeling using PyTorch/XLA on a TPUv3-8, and PyTorch on 8 V100 GPUs. We report the overall training time below. For reproducibility, we state the training commands used for PyTorch/XLA and PyTorch further below.

Task	TPU v3-8 (Flax)	TPU v3-8 (Pytorch/XLA)	8 GPU (PyTorch)
MLM	15h32m	23h46m	44h14m

*All experiments are ran on Google Cloud Platform. GPU experiments are ran without further optimizations besides JAX transformations. GPU experiments are ran with full precision (fp32). "TPU v3-8" are 8 TPU cores on 4 chips (each chips has 2 cores), while "8 GPU" are 8 GPU chips.

Script to run MLM with PyTorch/XLA on TPUv3-8

For comparison one can run the same pre-training with PyTorch/XLA on TPU. To set up PyTorch/XLA on Cloud TPU VMs, please refer to this guide. Having created the tokenzier and configuration in `norwegian-roberta-base`, we create the following symbolic links:

```
ln -s ~/transformers/examples/pytorch/language-modeling/run_mlm.py ./
ln -s ~/transformers/examples/pytorch/xla_spawn.py ./
```

, set the following environment variables:

```
export XRT_TPU_CONFIG="localservice;0;localhost:51011"
unset LD_PRELOAD

export NUM_TPUS=8
export TOKENIZERS_PARALLELISM=0
export MODEL_DIR="./norwegian-roberta-base"
mkdir -p ${MODEL_DIR}
```

, and start training as follows:

```
python3 xla_spawn.py --num_cores ${NUM_TPUS} run_mlm.py --output_dir="./runs" \
    --model_type="roberta" \
    --config_name="${MODEL_DIR}" \
    --tokenizer_name="${MODEL_DIR}" \
    --dataset_name="oscar" \
    --dataset_config_name="unshuffled_deduplicated_no" \
    --max_seq_length="128" \
    --weight_decay="0.01" \
    --per_device_train_batch_size="128" \
    --per_device_eval_batch_size="128" \
    --learning_rate="3e-4" \
    --warmup_steps="1000" \
    --overwrite_output_dir \
    --num_train_epochs="18" \
    --adam_beta1="0.9" \
    --adam_beta2="0.98" \
    --do_train \
    --do_eval \
    --logging_steps="500" \
    --evaluation_strategy="epoch" \
    --report_to="tensorboard" \
    --save_strategy="no"
```

Script to compare pre-training with PyTorch on 8 GPU V100's

For comparison you can run the same pre-training with PyTorch on GPU. Note that we have to make use of `gradient_accumulation` because the maximum batch size that fits on a single V100 GPU is 32 instead of 128. Having created the tokenzier and configuration in `norwegian-roberta-base`, we create the following symbolic links:

```
ln -s ~/transformers/examples/pytorch/language-modeling/run_mlm.py ./
```

, set some environment variables:

```
export NUM_GPUS=8
export TOKENIZERS_PARALLELISM=0
export MODEL_DIR="./norwegian-roberta-base"
mkdir -p ${MODEL_DIR}
```

, and can start training as follows:

```
python3 -m torch.distributed.launch --nproc_per_node ${NUM_GPUS} run_mlm.py \
    --output_dir="${MODEL_DIR}" \
    --model_type="roberta" \
    --config_name="${MODEL_DIR}" \
    --tokenizer_name="${MODEL_DIR}" \
    --dataset_name="oscar" \
    --dataset_config_name="unshuffled_deduplicated_no" \
    --max_seq_length="128" \
    --weight_decay="0.01" \
    --per_device_train_batch_size="32" \
    --per_device_eval_batch_size="32" \
    --gradient_accumulation="4" \
    --learning_rate="3e-4" \
    --warmup_steps="1000" \
    --overwrite_output_dir \
    --num_train_epochs="18" \
    --adam_beta1="0.9" \
    --adam_beta2="0.98" \
    --do_train \
    --do_eval \
    --logging_steps="500" \
    --evaluation_strategy="steps" \
    --report_to="tensorboard" \
    --save_strategy="no"
```