Introduction to Huggingface

Amin Saied

2020/18/11

Background

Transfer learning in NLP

- Transformers: **Large**, **high-capacity**
- Unsupervised learning: Language modelling

$$p_\theta(\boldsymbol{w}) \simeq \prod_{t=1}^{T} p_\theta(w_t|\boldsymbol{w}_{<t})$$

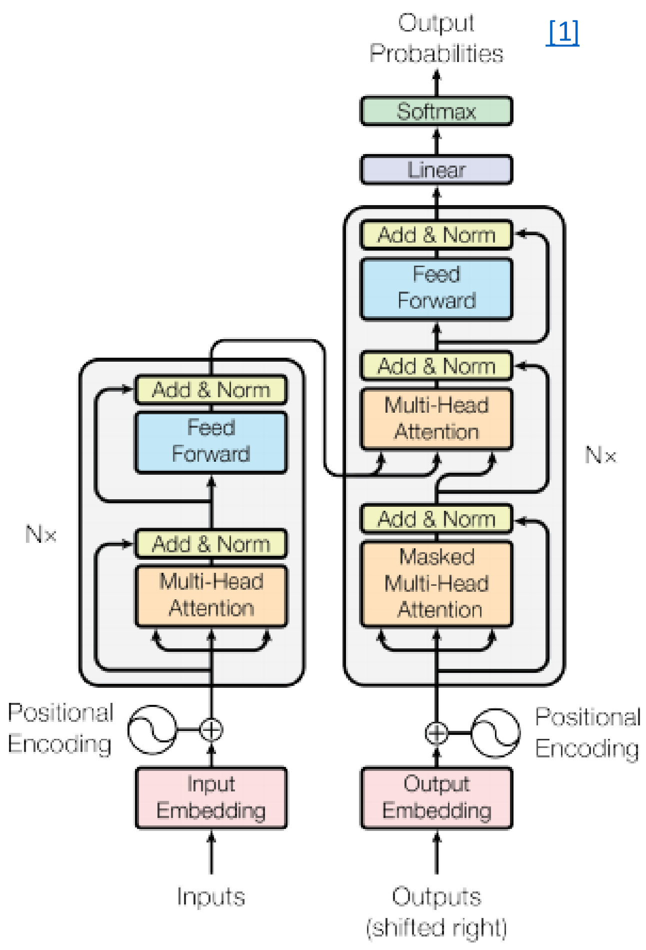
  - e.g. The best book ever written is <blank>
  - e.g. Masked Language Modelling: The capital of <blank> is Canberra.
  - e.g. Next Sentence Prediction:

    Many important tasks are based on next sentence prediction.

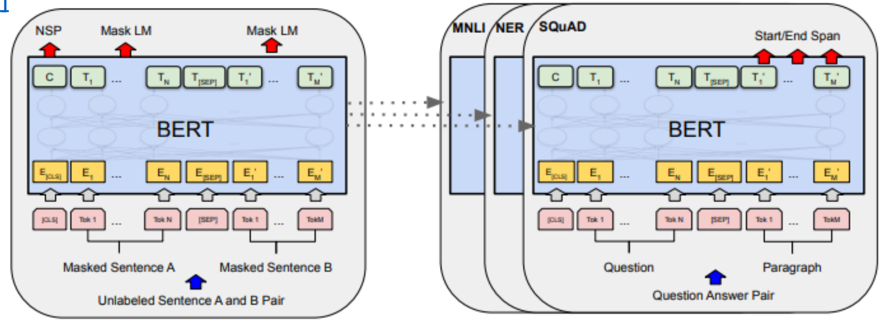
    Next add half a bottle of good quality red wine.

- Fine-tune: Transfer to specific task
  - e.g. Classification => "Put a linear layer on top"
  - Data efficient
  - SOTA performance

[1]



[2]



Pre-training Fine-Tuning

[1] Attention is all you need - Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser, Polosukhin

[2] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding - Devlin, Chang, Lee, Toutanova

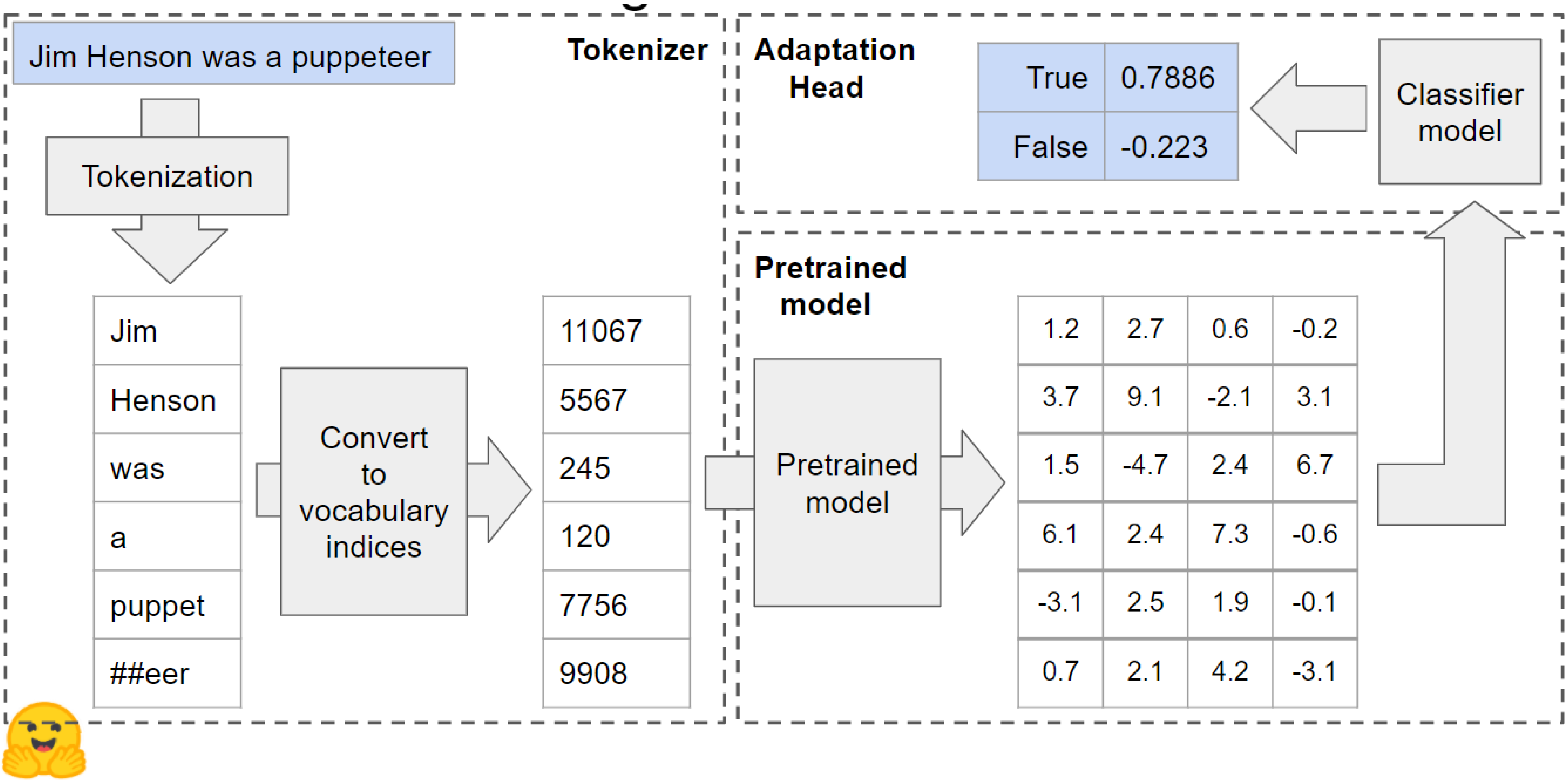
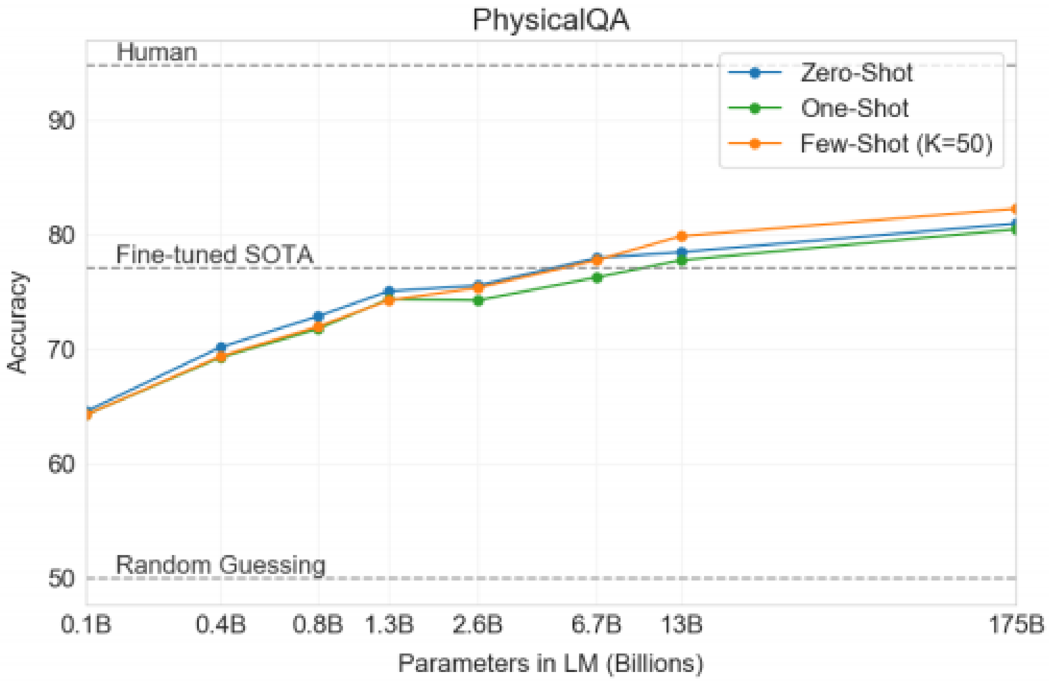


Image from: [Transfer Learning and HuggingFace - Thomas Wolf – CSO Huggingface]

Motivation

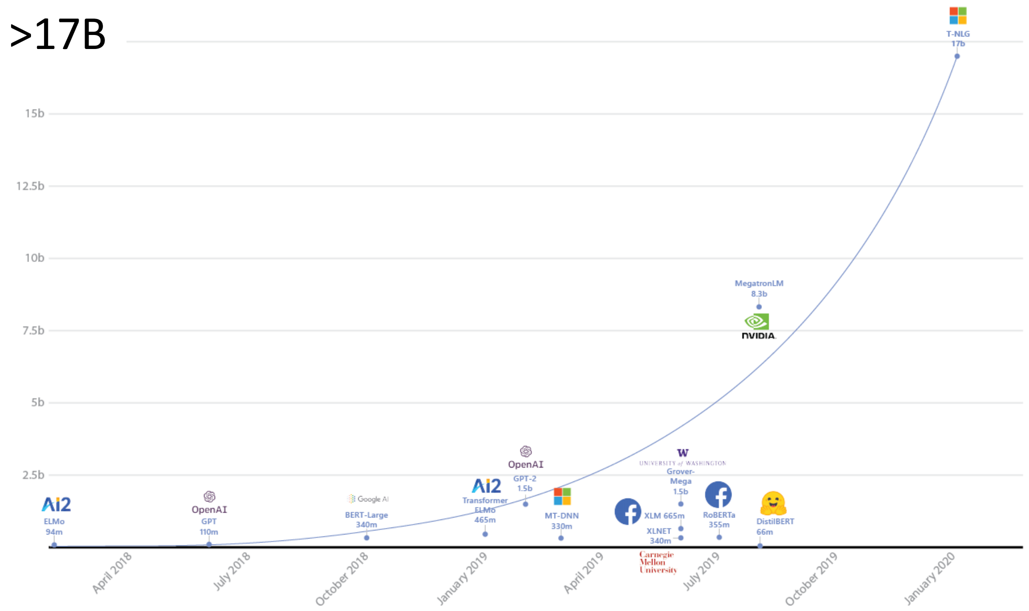
Bigger is better, still!

[Language models are few shot learners – Open AI]



Expensive to train, so share 🤗

[Turing-NLG: A 17-billion-parameter language model by Microsoft - MSR]



Share research / practitioners' techniques

- Distillation [3]
- Mixed precision [4]
- Distributed training (PyTorch docs)
- Gradient clipping [5]
- Quantization [6]
- …

[1] Language models are few shot learners – Open AI

[2] Turing-NLG: A 17-billion-parameter language model by Microsoft - MSR

[3] Distilling the knowledge in a neural network – Hinton, Vinyals, Dean

[4] Mixed precision training - Micikevicius Narang Alben Diamos Elsen Garcia Ginsburg Houston Kuchaiev Venkatesh Wu

[5] Why gradient clipping accelerated training, … - Zhang, He, Sra, Jadbabaie

[6] Faster and smaller quantized NLP with HugginFace and ONNX Runtime

🤍 The latest in Machine Learning | Papers With Code 🤍

Huggingface/transformers

Models / Tokenizers

- bert-base-uncased
- jplu/tf-xlm-roberta-base
- cl-tohoku/bert-base-japanese-whole-word-masking
- dccuchile/bert-base-spanish-wwm-cased
- distilbert-base-uncased
- microsoft/xprophetnet-large-wiki100-cased-xglue-ntg
- deepset/roberta-base-squad2
- roberta-base
- bert-base-cased
- xlm-roberta-base
- gpt2
- t5-small
- facebook/bart-large-mnli
- roberta-large
- valhalla/t5-small-qa-qg-hl
- bert-base-multilingual-cased
- valhalla/t5-small-qg-hl
- xlm-roberta-large-finetuned-conll03-german
- bert-large-cased-whole-word-masking
- bert-large-uncased
- albert-base-v2
- beomi/kcbert-base
- distilbert-base-uncased-finetuned-sst-2-english

Model

- Subclasses torch.nn.module
- Common methods e.g. for loading/saving
- Library of models (hosted on AWS S3)

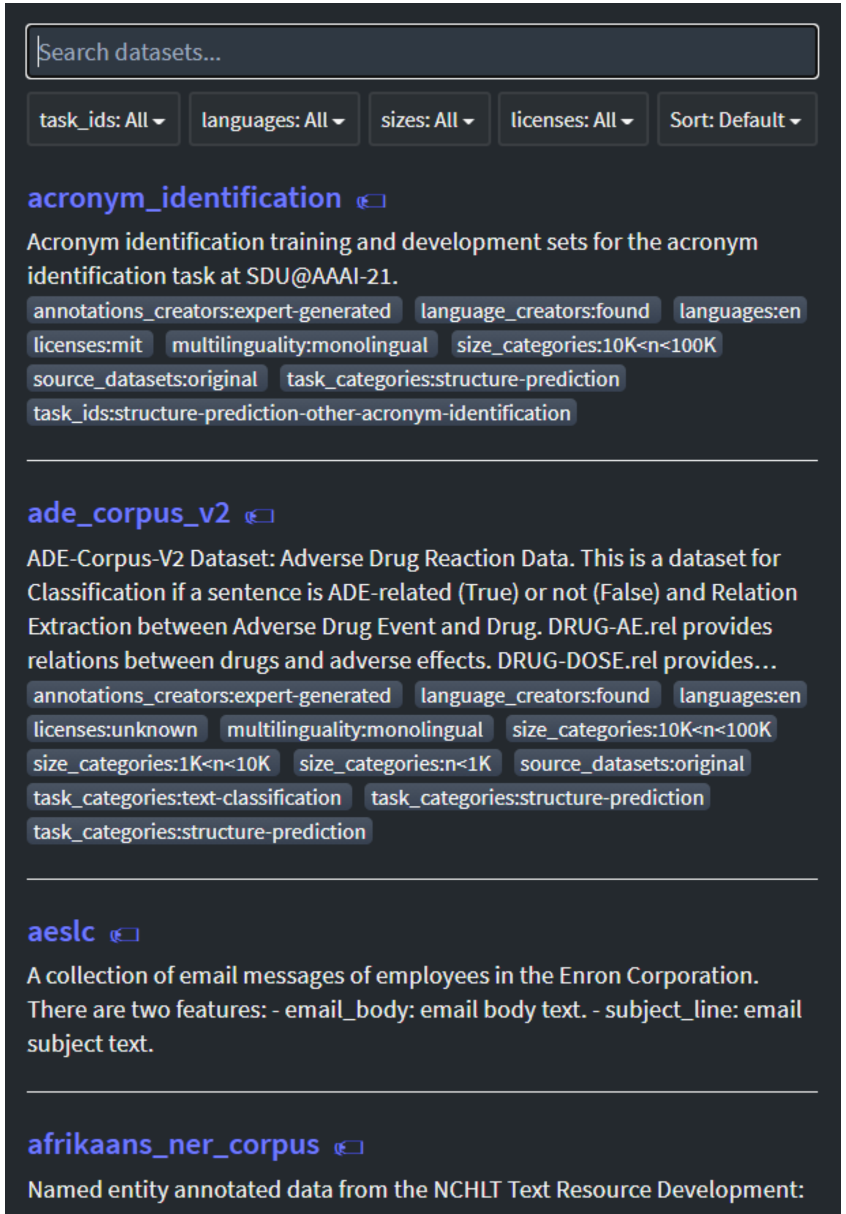
```python
from transformers import BertModel
model = BertModel.from_pretrained('bert-base-uncased')
```

Tokenizer (github/hf/tokenizers)

- Prepares the inputs for a model
- Coupled with specific models
- Encode / decode + padding / truncation

```python
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')
```

Datasets / Metrics



- Lightweight library for:
  - One-line dataloaders: >500 (and counting) major public datasets
  - Pre-processing
  - Metrics: Common API for dataset-specific metrics

```python
from datasets import list_datasets, load_dataset, list_metrics, load_metric

# Print all the available datasets
print(list_datasets())

# Load a dataset and print the first examples in the training set
squad_dataset = load_dataset('squad')
print(squad_dataset['train'][0])

# List all the available metrics
print(list_metrics())

# Load a metric
squad_metric = load_metric('squad')
```


```
{'answers': {'answer_start': [515], 'text': ['Saint Bernadette Soubirous']},
 'context': 'Architecturally, the school has a Catholic character. Atop the Main Building\'s gold dome is
nt of the Main Building and facing it, is a copper statue of Christ with arms upraised with the legend "Ve
silica of the Sacred Heart. Immediately behind the basilica is the Grotto, a Marian place of prayer and re
ance where the Virgin Mary reputedly appeared to Saint Bernadette Soubirous in 1858. At the end of the mai
statues and the Gold Dome), is a simple, modern stone statue of Mary.',
 'id': '5733be284776f41900661182',
 'question': 'To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France?',
 'title': 'University_of_Notre_Dame'}
```

Trainer

- Trainer is a **simple** but **feature-complete** training and eval loop optimized for 🤗 Transformers.
- Simple:
  - Black box approach
  - Interface:
    - Model
    - Tokenizer
    - Dataset
    - Metrics
    - Optimizer
  - Hooks: Callbacks
- Feature complete:
  - Distributed training
  - Mixedprecision training
  - Gradient accumulation
  - Gradient clipping
  - Learning rate scheduling
  - Save / load / checkpointing
  - Hyperparameter tuning

```python
parser = HfArgumentParser(TrainingArguments)
parser.add_argument("--task", default="cola", help="name of GLUE task to compute")
parser.add_argument("--model_checkpoint", default="distilbert-base-uncased")
training_args, args = parser.parse_args_into_dataclasses()

task: str = args.task.lower()

tokenizer = AutoTokenizer.from_pretrained(args.model_checkpoint, use_fast=True)

encoded_dataset_train, encoded_dataset_eval = load_encoded_glue_dataset(
    task=task, tokenizer=tokenizer
)

num_labels = num_labels_from_task(task)

model = AutoModelForSequenceClassification.from_pretrained(
    args.model_checkpoint, num_labels=num_labels
)

compute_metrics = construct_compute_metrics_function(args.task)

trainer = Trainer(
    model,
    training_args,
    callbacks=[AzureMLCallback()],
    train_dataset=encoded_dataset_train,
    eval_dataset=encoded_dataset_eval,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)

print("Training...")

run = Run.get_context()  # get handle on Azure ML run
start = time.time()
trainer.train()
run.log("time/epoch", (time.time() - start) / 60 / training_args.num_train_epochs)
```

TrainingArgs / HFArgumentParser

- Clean way to define arguments

```python
@dataclass
class TrainingArguments:
    """
    TrainingArguments is the subset of the arguments we use in our example scripts **which relate to the training loop
    itself**.
    """

    do_train: bool = field(default=False, metadata={"help": "Whether to run training."})
    do_eval: bool = field(default=None, metadata={"help": "Whether to run eval on the dev set."})
    model_parallel: bool = field(
        default=False,
        metadata={
            "help": (
                "If there are more than one devices, whether to use model parallelism to distribute the "
                "model's modules across devices."
            )
        },
    )

    learning_rate: float = field(default=5e-5, metadata={"help": "The initial learning rate for Adam."})
    weight_decay: float = field(default=0.0, metadata={"help": "Weight decay if we apply some."})
    adam_beta1: float = field(default=0.9, metadata={"help": "Beta1 for Adam optimizer"})
    adam_beta2: float = field(default=0.999, metadata={"help": "Beta2 for Adam optimizer"})
    adam_epsilon: float = field(default=1e-8, metadata={"help": "Epsilon for Adam optimizer."})
    max_grad_norm: float = field(default=1.0, metadata={"help": "Max gradient norm."})

    per_device_train_batch_size: int = field(
        default=8, metadata={"help": "Batch size per GPU/TPU core/CPU for training."}
    )
    per_device_eval_batch_size: int = field(
        default=8, metadata={"help": "Batch size per GPU/TPU core/CPU for evaluation."}
    )

    per_gpu_train_batch_size: Optional[int] = field(
        default=None,
        metadata={
            "help": "Deprecated, the use of `--per_device_train_batch_size` is preferred. "
            "Batch size per GPU/TPU core/CPU for training."
        },
    )

    gradient_accumulation_steps: int = field(
        default=1,
        metadata={"help": "Number of updates steps to accumulate before performing a backward/update pass."},
    )
```

...

```python
    def to_dict(self):
        """
        Serializes this instance while replace `Enum` by their values (for JSON serialization support).
        """
        d = dataclasses.asdict(self)
        for k, v in d.items():
            if isinstance(v, Enum):
                d[k] = v.value
        return d

    def to_json_string(self):
        """
        Serializes this instance to a JSON string.
        """
        return json.dumps(self.to_dict(), indent=2)
```

```python
class HfArgumentParser(ArgumentParser):
    """
    This subclass of `argparse.ArgumentParser` uses type hints on dataclasses to generate arguments.

    The class is designed to play well with the native argparse. In particular, you can add more (non-dataclass backed)
    arguments to the parser after initialization and you'll get the output back after parsing as an additional
    namespace.
    """

    dataclass_types: Iterable[DataClassType]

    def __init__(self, dataclass_types: Union[DataClassType, Iterable[DataClassType]], **kwargs):
        """
        Args:
            dataclass_types:
                Dataclass type, or list of dataclass types for which we will "fill" instances with the parsed args.
            kwargs:
                (Optional) Passed to `argparse.ArgumentParser()` in the regular way.
        """
        super().__init__(**kwargs)
        if dataclasses.is_dataclass(dataclass_types):
            dataclass_types = [dataclass_types]
        self.dataclass_types = dataclass_types
        for dtype in self.dataclass_types:
            self._add_dataclass_arguments(dtype)
```

```python
parser = HfArgumentParser(TrainingArguments)
parser.add_argument("--task", default="cola", help="name of GLUE task to compute")
parser.add_argument("--model_checkpoint", default="distilbert-base-uncased")
training_args, args = parser.parse_args_into_dataclasses()
```

```
train.py --learning_rate 3e-7 --weight_decay 0.2 --something_custom yes
```

Callbacks

- Customize behavior of training loop *without touching trainer*
- Inspect training state (e.g. progress reporting, TensorBoard…)
- Take decisions (e.g. early stopping)
- Defaults:
  - DefaultFlowCallback
  - PrinterCallback / ProgressCallback
  - TensorBoardCallback
  - AzureMLCallback
- Customizable!
  - on_epoch_begin, on_epoch_end, on_evaluate, on_init_end, on_log, on_prediction_step, on_save, on_step_begin, on_step_end, on_train_begin, on_train_end
- Example: EarlyStoppingCallback

```python
class PrinterCallback(TrainerCallback):

    def on_log(self, args, state, control, logs=None, **kwargs):
        _ = logs.pop("total_flos", None)
        if state.is_local_process_zero:
            print(logs)
```

```python
trainer = Trainer(
    model,
    training_args,
    callbacks=[AzureMLCallback()],
    train_dataset=encoded_dataset_train,
    eval_dataset=encoded_dataset_eval,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
)

print("Training...")

run = Run.get_context()  # get handle on Azure ML run
start = time.time()
trainer.train()
run.log("time/epoch", (time.time() - start) / 60 / training_args.num_train_epochs)
```

NB. Only on PyTorch for now

Demo

Demos and examples

- Example notebook (simple)*
- Azureml-examples repo**
    - GLUE finetuning, based on official HF example

* TODO: share notebook in aml-ds repo
** Not yet on master (amsaied/workflows/transformers)