NLP_Sujet2_Part1

November 23, 2021

1 Finetunes models that answer question by taking a substring of a context.

---

2 Imports

[1]:
```
! pip install datasets transformers
```

```
Collecting datasets
  Downloading datasets-1.15.1-py3-none-any.whl (290 kB)
     || 290 kB 5.4 MB/s
Collecting transformers
  Downloading transformers-4.12.5-py3-none-any.whl (3.1 MB)
     || 3.1 MB 31.4 MB/s
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7
/dist-packages (from datasets) (1.19.5)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages
(from datasets) (1.1.5)
Collecting xxhash
  Downloading xxhash-2.0.2-cp37-cp37m-manylinux2010_x86_64.whl (243 kB)
     || 243 kB 37.8 MB/s
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.7
/dist-packages (from datasets) (4.62.3)
Requirement already satisfied: dill in /usr/local/lib/python3.7/dist-packages
(from datasets) (0.3.4)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.7/dist-
packages (from datasets) (0.70.12.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-
packages (from datasets) (21.3)
Collecting fsspec[http]>=2021.05.0
  Downloading fsspec-2021.11.0-py3-none-any.whl (132 kB)
     || 132 kB 37.5 MB/s
Collecting huggingface-hub<1.0.0,>=0.1.0
  Downloading huggingface_hub-0.1.2-py3-none-any.whl (59 kB)
```

```
     || 59 kB 5.2 MB/s
Collecting aiohttp
  Downloading aiohttp-3.8.1-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.ma
nylinux_2_12_x86_64.manylinux2010_x86_64.whl (1.1 MB)
     || 1.1 MB 35.2 MB/s
Requirement already satisfied: importlib-metadata in
/usr/local/lib/python3.7/dist-packages (from datasets) (4.8.2)
Requirement already satisfied: pyarrow!=4.0.0,>=1.0.0 in
/usr/local/lib/python3.7/dist-packages (from datasets) (3.0.0)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.7
/dist-packages (from datasets) (2.23.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages
(from huggingface-hub<1.0.0,>=0.1.0->datasets) (3.13)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.7/dist-packages (from huggingface-
hub<1.0.0,>=0.1.0->datasets) (3.10.0.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-
packages (from huggingface-hub<1.0.0,>=0.1.0->datasets) (3.4.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from packaging->datasets) (3.0.6)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7
/dist-packages (from requests>=2.19.0->datasets) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.19.0->datasets)
(1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
/dist-packages (from requests>=2.19.0->datasets) (2021.10.8)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests>=2.19.0->datasets) (2.10)
Collecting tokenizers<0.11,>=0.10.1
  Downloading tokenizers-0.10.3-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_6
4.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (3.3 MB)
     || 3.3 MB 37.9 MB/s
Collecting sacremoses
  Downloading sacremoses-0.0.46-py3-none-any.whl (895 kB)
     || 895 kB 48.8 MB/s
Collecting pyyaml
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manyl
inux_2_12_x86_64.manylinux2010_x86_64.whl (596 kB)
     || 596 kB 47.4 MB/s
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.7/dist-packages (from transformers) (2019.12.20)
Collecting async-timeout<5.0,>=4.0.0a3
  Downloading async_timeout-4.0.1-py3-none-any.whl (5.7 kB)
Collecting asynctest==0.13.0
  Downloading asynctest-0.13.0-py3-none-any.whl (26 kB)
Requirement already satisfied: charset-normalizer<3.0,>=2.0 in
/usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (2.0.7)
```

```
Collecting yarl<2.0,>=1.0
  Downloading yarl-1.7.2-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manyl
inux_2_12_x86_64.manylinux2010_x86_64.whl (271 kB)
     || 271 kB 39.6 MB/s
Collecting multidict<7.0,>=4.5
  Downloading multidict-5.2.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.
manylinux_2_12_x86_64.manylinux2010_x86_64.whl (160 kB)
     || 160 kB 31.4 MB/s
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.7
/dist-packages (from aiohttp->datasets) (21.2.0)
Collecting aiosignal>=1.1.2
  Downloading aiosignal-1.2.0-py3-none-any.whl (8.2 kB)
Collecting frozenlist>=1.1.1
  Downloading frozenlist-1.2.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64
.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (192 kB)
     || 192 kB 48.5 MB/s
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-metadata->datasets) (3.6.0)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.7/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
packages (from pandas->datasets) (2018.9)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-
packages (from python-dateutil>=2.7.3->pandas->datasets) (1.15.0)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers) (7.1.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers) (1.1.0)
Installing collected packages: multidict, frozenlist, yarl, asynctest, async-
timeout, aiosignal, pyyaml, fsspec, aiohttp, xxhash, tokenizers, sacremoses,
huggingface-hub, transformers, datasets
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed aiohttp-3.8.1 aiosignal-1.2.0 async-timeout-4.0.1
asynctest-0.13.0 datasets-1.15.1 frozenlist-1.2.0 fsspec-2021.11.0 huggingface-
hub-0.1.2 multidict-5.2.0 pyyaml-6.0 sacremoses-0.0.46 tokenizers-0.10.3
transformers-4.12.5 xxhash-2.0.2 yarl-1.7.2
```

[2]:
```python
from datasets import load_dataset, load_metric
import transformers
from transformers import AutoTokenizer
from transformers import AutoModelForQuestionAnswering, TrainingArguments,␣
 ↪Trainer
from transformers import default_data_collator
import collections
```

```python
from tqdm.auto import tqdm
import numpy as np
```

3 Parameters

[3]:
```python
batch_size = 16
epochs = 3

max_length = 384 # The maximum length of a feature (question and context)
doc_stride = 128 # The authorized overlap between two part of the context when␣
 ↪splitting it is needed.
```

4 Load Data

[4]:
```python
#Load train and validation dataset
datasets = load_dataset("squad_v2")
```

```
Downloading:   0%|          | 0.00/1.87k [00:00<?, ?B/s]


Downloading:   0%|          | 0.00/1.02k [00:00<?, ?B/s]


Downloading and preparing dataset squad_v2/squad_v2 (download: 44.34 MiB,
generated: 122.41 MiB, post-processed: Unknown size, total: 166.75 MiB) to /root
/.cache/huggingface/datasets/squad_v2/squad_v2/2.0.0/09187c73c1b837c95d9a249cd97
c2c3f1cebada06efe667b4427714b27639b1d...

  0%|          | 0/2 [00:00<?, ?it/s]


Downloading:   0%|          | 0.00/9.55M [00:00<?, ?B/s]


Downloading:   0%|          | 0.00/801k [00:00<?, ?B/s]


  0%|          | 0/2 [00:00<?, ?it/s]


0 examples [00:00, ? examples/s]


0 examples [00:00, ? examples/s]
```

```
Dataset squad_v2 downloaded and prepared to /root/.cache/huggingface/datasets/sq
uad_v2/squad_v2/2.0.0/09187c73c1b837c95d9a249cd97c2c3f1cebada06efe667b4427714b27
639b1d. Subsequent calls will reuse this data.
```

```
  0%|          | 0/2 [00:00<?, ?it/s]
```

[5]:
```
datasets
```

[5]:
```
DatasetDict({
    train: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 130319
    })
    validation: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 11873
    })
})
```

All datasets have a column for the context, the question and the answers to those questions

[6]:
```
datasets["train"][0]
```

[6]:
```
{'answers': {'answer_start': [269], 'text': ['in the late 1990s']},
 'context': 'Beyoncé Giselle Knowles-Carter (/bijnse/ bee-YON-say) (born
September 4, 1981) is an American singer, songwriter, record producer and
actress. Born and raised in Houston, Texas, she performed in various singing and
dancing competitions as a child, and rose to fame in the late 1990s as lead
singer of R&B girl-group Destiny\'s Child. Managed by her father, Mathew
Knowles, the group became one of the world\'s best-selling girl groups of all
time. Their hiatus saw the release of Beyoncé\'s debut album, Dangerously in
Love (2003), which established her as a solo artist worldwide, earned five
Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in
Love" and "Baby Boy".',
 'id': '56be85543aeaaa14008c9063',
 'question': 'When did Beyonce start becoming popular?',
 'title': 'Beyoncé'}
```

We can see above the answers are indicated by their start position in the text (here at character 269).

[7]:
```
# Shuffle training set and use a subset of training samples

datasets["train"] = datasets["train"].shuffle(seed=1)
datasets["train"] = datasets["train"].shard(num_shards=25, index=0) #5273␣
 ↪samples

# Reduce number of samples for testing our code #TODO
datasets["validation"] = datasets["validation"].shuffle(seed=1)
datasets["validation"] = datasets["validation"].shard(num_shards=2, index=0)␣
 ↪#6072 samples
```

5 Preprocesing the training data

[8]:
```
# This function works with one or several examples. In the case of several␣
↪examples, the tokenizer will return a list of lists for each ke
def prepare_train_features(examples, tokenizer):
    pad_on_right = tokenizer.padding_side == "right"
    # Some of the questions have lots of whitespace on the left, which is not␣
↪useful and will make the
    # truncation of the context fail (the tokenized question will take a lots␣
↪of space). So we remove that
    # left whitespace
    examples["question"] = [q.lstrip() for q in examples["question"]]

    # Tokenize our examples with truncation and padding, but keep the overflows␣
↪using a stride. This results
    # in one example possible giving several features when a context is long,␣
↪each of those features having a
    # context that overlaps a bit the context of the previous feature.
    tokenized_examples = tokenizer(
        examples["question" if pad_on_right else "context"],
        examples["context" if pad_on_right else "question"],
        truncation="only_second" if pad_on_right else "only_first",
        max_length=max_length,
        stride=doc_stride,
        return_overflowing_tokens=True,
        return_offsets_mapping=True,
        padding="max_length",
    )

    # Since one example might give us several features if it has a long␣
↪context, we need a map from a feature to
    # its corresponding example. This key gives us just that.
    sample_mapping = tokenized_examples.pop("overflow_to_sample_mapping")
    # The offset mappings will give us a map from token to character position␣
↪in the original context. This will
    # help us compute the start_positions and end_positions.
    offset_mapping = tokenized_examples.pop("offset_mapping")

    # Let's label those examples!
    tokenized_examples["start_positions"] = []
    tokenized_examples["end_positions"] = []

    for i, offsets in enumerate(offset_mapping):
        # We will label impossible answers with the index of the CLS token.
        input_ids = tokenized_examples["input_ids"][i]
        cls_index = input_ids.index(tokenizer.cls_token_id)
        # Grab the sequence corresponding to that example (to know what is the␣
↪context and what is the question).
        sequence_ids = tokenized_examples.sequence_ids(i)

        # One example can give several spans, this is the index of the example␣
↪containing this span of text.
        sample_index = sample_mapping[i]
        answers = examples["answers"][sample_index]
        # If no answers are given, set the cls_index as answer.
        if len(answers["answer_start"]) == 0:
            tokenized_examples["start_positions"].append(cls_index)
            tokenized_examples["end_positions"].append(cls_index)
        else:
            # Start/end character index of the answer in the text.
            start_char = answers["answer_start"][0]
            end_char = start_char + len(answers["text"][0])

            # Start token index of the current span in the text.
            token_start_index = 0
            while sequence_ids[token_start_index] != (1 if pad_on_right else 0):
                token_start_index += 1

            # End token index of the current span in the text.
            token_end_index = len(input_ids) - 1
            while sequence_ids[token_end_index] != (1 if pad_on_right else 0):
                token_end_index -= 1

            # Detect if the answer is out of the span (in which case this␣
↪feature is labeled with the CLS index).
            if not (offsets[token_start_index][0] <= start_char and␣
↪offsets[token_end_index][1] >= end_char):
                tokenized_examples["start_positions"].append(cls_index)
                tokenized_examples["end_positions"].append(cls_index)
            else:
                # Otherwise move the token_start_index and token_end_index to␣
↪the two ends of the answer.
                # Note: we could go after the last offset if the answer is the␣
↪last word (edge case).
                while token_start_index < len(offsets) and␣
↪offsets[token_start_index][0] <= start_char:
                    token_start_index += 1
                tokenized_examples["start_positions"].append(token_start_index␣
↪- 1)
                while offsets[token_end_index][1] >= end_char:
                    token_end_index -= 1
                tokenized_examples["end_positions"].append(token_end_index + 1)

    return tokenized_examples
```

[9]:
```
def pre_process(datasets, tokenizer):
  '''
   Apply the 'prepare_train_features' on all the elements of all the splits in␣
↪dataset,
   so our training, validation and testing data will be preprocessed in one␣
↪single command.
   Since our preprocessing changes the number of samples, we need to remove the␣
↪old columns when applying it.

   batched=True to encode the texts by batches together.
   This is to leverage the full benefit of the fast tokenizer we loaded␣
↪earlier,
   which will use multi-threading to treat the texts in a batch concurrently.

  '''
  tokenized_datasets = datasets.map(lambda x: prepare_train_features(x,␣
↪tokenizer), batched=True, remove_columns=datasets["train"].column_names)
  return tokenized_datasets
```

6 Fine-tuning the model

Now that our data is ready for training, we can download the pretrained model and fine-tune it. Since our task is question answering, we use the AutoModelForQuestionAnswering class. Like with the tokenizer, the from_pretrained method will download and cache the model for us:

[10]:
```
def produce_trainer(model, args, tokenizer, tokenized_datasets):
  # A data collator will batch our processed examples together
  data_collator = default_data_collator
  trainer = Trainer(
      model,
      args,
      train_dataset=tokenized_datasets["train"],
      eval_dataset=tokenized_datasets["validation"],
      data_collator=data_collator,
      tokenizer=tokenizer,
  )
  return trainer
```

7 Evaluation

The output of the model is a dict-like object that contains the loss (since we provided labels), the start and end logits. \

To classify our answers, we will use the score obtained by adding the start and end logits. We won't try to order all the possible answers and limit ourselves to with a hyper-parameter we call n_best_size. We'll pick the best indices in the start and end logits and gather all the answers this predicts. After checking if each one is valid, we will sort them by their score and keep the best one.

[11]:
```
#  re-process the validation set with the following function

def prepare_validation_features(examples, tokenizer):
    pad_on_right = tokenizer.padding_side == "right"
    # Some of the questions have lots of whitespace on the left, which is not␣
↪useful and will make the
    # truncation of the context fail (the tokenized question will take a lots␣
↪of space). So we remove that
    # left whitespace
    examples["question"] = [q.lstrip() for q in examples["question"]]

    # Tokenize our examples with truncation and maybe padding, but keep the␣
↪overflows using a stride. This results
    # in one example possible giving several features when a context is long,␣
↪each of those features having a
    # context that overlaps a bit the context of the previous feature.
    tokenized_examples = tokenizer(
        examples["question" if pad_on_right else "context"],
        examples["context" if pad_on_right else "question"],
        truncation="only_second" if pad_on_right else "only_first",
        max_length=max_length,
        stride=doc_stride,
        return_overflowing_tokens=True,
        return_offsets_mapping=True,
        padding="max_length",
    )

    # Since one example might give us several features if it has a long␣
↪context, we need a map from a feature to
    # its corresponding example. This key gives us just that.
    sample_mapping = tokenized_examples.pop("overflow_to_sample_mapping")

    # We keep the example_id that gave us this feature and we will store the␣
↪offset mappings.
    tokenized_examples["example_id"] = []

    for i in range(len(tokenized_examples["input_ids"])):
        # Grab the sequence corresponding to that example (to know what is the␣
↪context and what is the question).
        sequence_ids = tokenized_examples.sequence_ids(i)
        context_index = 1 if pad_on_right else 0
        # One example can give several spans, this is the index of the example␣
↪containing this span of text.
        sample_index = sample_mapping[i]
        tokenized_examples["example_id"].append(examples["id"][sample_index])

        # Set to None the offset_mapping that are not part of the context so␣
↪it's easy to determine if a token
        # position is part of the context or not.
        tokenized_examples["offset_mapping"][i] = [
            (o if sequence_ids[k] == context_index else None)
            for k, o in enumerate(tokenized_examples["offset_mapping"][i])
        ]

    return tokenized_examples
```

[12]:
```
# Apply "prepare_validation_features" to our validation set
def map_validation_features(tokenizer):
  validation_features = datasets["validation"].map(lambda x:␣
↪prepare_validation_features(x, tokenizer),
      batched=True,
      remove_columns=datasets["validation"].column_names
  )
  return validation_features
```

[13]:
```
def postprocess_qa_predictions(tokenizer, examples, features, raw_predictions,␣
↪n_best_size = 20, max_answer_length = 30):
    all_start_logits, all_end_logits = raw_predictions
    # Build a map example to its corresponding features.
    example_id_to_index = {k: i for i, k in enumerate(examples["id"])}
    features_per_example = collections.defaultdict(list)
    for i, feature in enumerate(features):
        features_per_example[example_id_to_index[feature["example_id"]]].
↪append(i)

    # The dictionaries we have to fill.
    predictions = collections.OrderedDict()

    # Logging.
    print(f"Post-processing {len(examples)} example predictions split into␣
↪{len(features)} features.")

    # Let's loop over all the examples!
    for example_index, example in enumerate(tqdm(examples)):
        # Those are the indices of the features associated to the current␣
↪example.
        feature_indices = features_per_example[example_index]
        min_null_score = None # Only used if squad_v2 is True.
        valid_answers = []

        context = example["context"]
        # Looping through all the features associated to the current example.
        for feature_index in feature_indices:
            # We grab the predictions of the model for this feature.
            start_logits = all_start_logits[feature_index]
            end_logits = all_end_logits[feature_index]
            # This is what will allow us to map some the positions in our␣
↪logits to span of texts in the original
            # context.
            offset_mapping = features[feature_index]["offset_mapping"]

            # Update minimum null prediction.
            cls_index = features[feature_index]["input_ids"].index(tokenizer.
↪cls_token_id)
            feature_null_score = start_logits[cls_index] + end_logits[cls_index]
            if min_null_score is None or min_null_score < feature_null_score:
                min_null_score = feature_null_score

            # Go through all possibilities for the `n_best_size` greater start␣
↪and end logits.
            start_indexes = np.argsort(start_logits)[-1 : -n_best_size - 1 :␣
↪-1].tolist()
            end_indexes = np.argsort(end_logits)[-1 : -n_best_size - 1 : -1].
↪tolist()
            for start_index in start_indexes:
                for end_index in end_indexes:
                    # Don't consider out-of-scope answers, either because the␣
↪indices are out of bounds or correspond
                    # to part of the input_ids that are not in the context.
                    if (
                        start_index >= len(offset_mapping)
                        or end_index >= len(offset_mapping)
                        or offset_mapping[start_index] is None
                        or offset_mapping[end_index] is None
                    ):
                        continue
                    # Don't consider answers with a length that is either < 0␣
↪or > max_answer_length.
                    if end_index < start_index or end_index - start_index + 1 >␣
↪max_answer_length:
                        continue

                    start_char = offset_mapping[start_index][0]
                    end_char = offset_mapping[end_index][1]
                    valid_answers.append(
                        {
                            "score": start_logits[start_index] +␣
↪end_logits[end_index],
                            "text": context[start_char: end_char]
                        }
                    )

        if len(valid_answers) > 0:
            best_answer = sorted(valid_answers, key=lambda x: x["score"],␣
↪reverse=True)[0]
        else:
            # In the very rare edge case we have not a single non-null␣
↪prediction, we create a fake prediction to avoid
            # failure.
            best_answer = {"text": "", "score": 0.0}

        # Let's pick our final answer: the best one or the null answer (only␣
↪for squad_v2)
        answer = best_answer["text"] if best_answer["score"] > min_null_score␣
↪else ""
        predictions[example["id"]] = answer

    return predictions
```

[14]:
```
def evaluation(datasets, model, tokenizer, trainer):
    validation_features = map_validation_features(tokenizer)

    # Grab the predictions for all feature
    raw_predictions = trainer.predict(validation_features)

    # The Trainer hides the columns that are not used by the model
    # (that we will need for our post-processing), so we set them back
    validation_features.set_format(type=validation_features.format["type"],␣
↪columns=list(validation_features.features.keys()))

    # Eliminate very long answers from our considerations
    max_answer_length = 30

    # Build a map from example index to its corresponding features indices:
    examples = datasets["validation"]
    features = validation_features

    example_id_to_index = {k: i for i, k in enumerate(examples["id"])}
    features_per_example = collections.defaultdict(list)
    for i, feature in enumerate(features):
```

```
    features_per_example[example_id_to_index[feature["example_id"]]].append(i)

  final_predictions = postprocess_qa_predictions(tokenizer,␣
↪datasets["validation"], validation_features, raw_predictions.predictions)

  formatted_predictions = [{"id": k, "prediction_text": v,␣
↪"no_answer_probability": 0.0} for k, v in final_predictions.items()]
  references = [{"id": ex["id"], "answers": ex["answers"]} for ex in␣
↪datasets["validation"]]

  return formatted_predictions, references
```

[15]:
```
def process(datasets, tokenizer, model, args):
  tokenized_datasets = pre_process(datasets, tokenizer)
  trainer = produce_trainer(model, args, tokenizer, tokenized_datasets)
  # Finetuning our model
  trainer.train()
  # Since this training is particularly long, let's save the model
  #trainer.save_model("squad2-trained")
  # Evaluation
  preds, refs = evaluation(datasets, model, tokenizer, trainer)

  return preds, refs
```

7.1 Test on distilbert-base-uncased model

[16]:
```
# Model
model_checkpoint = "distilbert-base-uncased"
model1 = AutoModelForQuestionAnswering.from_pretrained(model_checkpoint)
# Tokenizer
tokenizer1 = AutoTokenizer.from_pretrained(model_checkpoint)


# The TrainingArguments is a class that contains all the attributes to␣
↪customize the training.
# The folder name will be used to save the checkpoints of the model
model_name1 = model_checkpoint.split("/")[-1]
args1 = TrainingArguments(
    f"{model_name1}-finetuned-squad",
    evaluation_strategy = "epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=epochs,
    weight_decay=0.01,
)
```

```
formatted_predictions1, references1 = process(datasets, tokenizer1, model1,␣
↪args1)
metric = load_metric("squad_v2")
```

```
Downloading:   0%|          | 0.00/483 [00:00<?, ?B/s]


Downloading:   0%|          | 0.00/256M [00:00<?, ?B/s]


Some weights of the model checkpoint at distilbert-base-uncased were not used
when initializing DistilBertForQuestionAnswering: ['vocab_layer_norm.bias',
'vocab_transform.weight', 'vocab_transform.bias', 'vocab_layer_norm.weight',
'vocab_projector.bias', 'vocab_projector.weight']
- This IS expected if you are initializing DistilBertForQuestionAnswering from
the checkpoint of a model trained on another task or with another architecture
(e.g. initializing a BertForSequenceClassification model from a
BertForPreTraining model).
- This IS NOT expected if you are initializing DistilBertForQuestionAnswering
from the checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
Some weights of DistilBertForQuestionAnswering were not initialized from the
model checkpoint at distilbert-base-uncased and are newly initialized:
['qa_outputs.bias', 'qa_outputs.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Downloading:   0%|          | 0.00/28.0 [00:00<?, ?B/s]


Downloading:   0%|          | 0.00/226k [00:00<?, ?B/s]


Downloading:   0%|          | 0.00/455k [00:00<?, ?B/s]


  0%|          | 0/6 [00:00<?, ?ba/s]


  0%|          | 0/6 [00:00<?, ?ba/s]


***** Running training *****
  Num examples = 5273
  Num Epochs = 3
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 16
  Gradient Accumulation steps = 1
  Total optimization steps = 990
```

```
<IPython.core.display.HTML object>


***** Running Evaluation *****
  Num examples = 6072
  Batch size = 16
Saving model checkpoint to distilbert-base-uncased-finetuned-
squad/checkpoint-500
Configuration saved in distilbert-base-uncased-finetuned-
squad/checkpoint-500/config.json
Model weights saved in distilbert-base-uncased-finetuned-
squad/checkpoint-500/pytorch_model.bin
tokenizer config file saved in distilbert-base-uncased-finetuned-
squad/checkpoint-500/tokenizer_config.json
Special tokens file saved in distilbert-base-uncased-finetuned-
squad/checkpoint-500/special_tokens_map.json
***** Running Evaluation *****
  Num examples = 6072
  Batch size = 16
***** Running Evaluation *****
  Num examples = 6072
  Batch size = 16


Training completed. Do not forget to share your model on huggingface.co/models
=)



  0%|          | 0/6 [00:00<?, ?ba/s]


The following columns in the test set  don't have a corresponding argument in
`DistilBertForQuestionAnswering.forward` and have been ignored: offset_mapping,
example_id.
***** Running Prediction *****
  Num examples = 6072
  Batch size = 16

<IPython.core.display.HTML object>


Post-processing 5937 example predictions split into 6072 features.

  0%|          | 0/5937 [00:00<?, ?it/s]


Downloading:   0%|          | 0.00/2.26k [00:00<?, ?B/s]
```

```
Downloading:   0%|          | 0.00/3.18k [00:00<?, ?B/s]
```

[17]:
```
metric.compute(predictions=formatted_predictions1, references=references1)
```

[17]:
```
{'HasAns_exact': 31.59675236806495,
 'HasAns_f1': 36.900098704626316,
 'HasAns_total': 2956,
 'NoAns_exact': 59.71150620597115,
 'NoAns_f1': 59.71150620597115,
 'NoAns_total': 2981,
 'best_exact': 50.27791814047499,
 'best_exact_thresh': 0.0,
 'best_f1': 50.36720472152805,
 'best_f1_thresh': 0.0,
 'exact': 45.713323227219135,
 'f1': 48.353830515559274,
 'total': 5937}
```

7.2 Test on mvonwyl/distilbert-base-uncased-finetuned-squad2 model

[19]:
```
model2 = AutoModelForQuestionAnswering.from_pretrained("mvonwyl/
 ↪distilbert-base-uncased-finetuned-squad2")
tokenizer2 = AutoTokenizer.from_pretrained("mvonwyl/
 ↪distilbert-base-uncased-finetuned-squad2")

model_name2 = "test_model_prof"
args2 = TrainingArguments(f"{model_name2}-finetuned-squad")


formatted_predictions2, references2 = process(datasets, tokenizer2, model2,␣
 ↪args2)
metric = load_metric("squad_v2")
```

```
https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/config.json not found in cache or force_download set to
True, downloading to /root/.cache/huggingface/transformers/tmpppradikc

Downloading:   0%|          | 0.00/561 [00:00<?, ?B/s]


storing https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/config.json in cache at /root/.cache/huggingface/transformer
s/cffbca5d7cd141d50b708ec6a05c79978d19c4dd3074fcd116c149cd355221f8.a540da103e8b3
d40c8db787d1fe802d9215f260969fa27deaa13b88795d8181b
creating metadata file for /root/.cache/huggingface/transformers/cffbca5d7cd141d
50b708ec6a05c79978d19c4dd3074fcd116c149cd355221f8.a540da103e8b3d40c8db787d1fe802
d9215f260969fa27deaa13b88795d8181b
```

```
loading configuration file https://huggingface.co/mvonwyl/distilbert-base-
uncased-finetuned-squad2/resolve/main/config.json from cache at /root/.cache/hug
gingface/transformers/cffbca5d7cd141d50b708ec6a05c79978d19c4dd3074fcd116c149cd35
5221f8.a540da103e8b3d40c8db787d1fe802d9215f260969fa27deaa13b88795d8181b
Model config DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForQuestionAnswering"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "torch_dtype": "float32",
  "transformers_version": "4.12.5",
  "vocab_size": 30522
}

https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/pytorch_model.bin not found in cache or force_download set
to True, downloading to /root/.cache/huggingface/transformers/tmp43hysxv5

Downloading:   0%|          | 0.00/253M [00:00<?, ?B/s]


storing https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/pytorch_model.bin in cache at /root/.cache/huggingface/trans
formers/b5b35581fa082df7d2ff2451f726fd2ebd8be192e2a873d4f7e53285f94db045.19e0e53
a6167116b32114f34d67c198cfdaa42db408fe90a32add525213a6fc1
creating metadata file for /root/.cache/huggingface/transformers/b5b35581fa082df
7d2ff2451f726fd2ebd8be192e2a873d4f7e53285f94db045.19e0e53a6167116b32114f34d67c19
8cfdaa42db408fe90a32add525213a6fc1
loading weights file https://huggingface.co/mvonwyl/distilbert-base-uncased-
finetuned-squad2/resolve/main/pytorch_model.bin from cache at /root/.cache/huggi
ngface/transformers/b5b35581fa082df7d2ff2451f726fd2ebd8be192e2a873d4f7e53285f94d
b045.19e0e53a6167116b32114f34d67c198cfdaa42db408fe90a32add525213a6fc1
All model checkpoint weights were used when initializing
```

```
DistilBertForQuestionAnswering.

All the weights of DistilBertForQuestionAnswering were initialized from the
model checkpoint at mvonwyl/distilbert-base-uncased-finetuned-squad2.
If your task is similar to the task the model of the checkpoint was trained on,
you can already use DistilBertForQuestionAnswering for predictions without
further training.
https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/tokenizer_config.json not found in cache or force_download
set to True, downloading to /root/.cache/huggingface/transformers/tmpewa2fqji

Downloading:   0%|          | 0.00/333 [00:00<?, ?B/s]


storing https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/tokenizer_config.json in cache at /root/.cache/huggingface/t
ransformers/0eefcbdecb44adc8a8a31203c6fc6081e4abee730e80b3d7c73ae8149d76268e.421
54c5fd30bfa7e34941d0d8ad26f8a3936990926fbe06b2da76dd749b1c6d4
creating metadata file for /root/.cache/huggingface/transformers/0eefcbdecb44adc
8a8a31203c6fc6081e4abee730e80b3d7c73ae8149d76268e.42154c5fd30bfa7e34941d0d8ad26f
8a3936990926fbe06b2da76dd749b1c6d4
https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/vocab.txt not found in cache or force_download set to True,
downloading to /root/.cache/huggingface/transformers/tmp9wtsdwxw

Downloading:   0%|          | 0.00/226k [00:00<?, ?B/s]


storing https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/vocab.txt in cache at /root/.cache/huggingface/transformers/
571aebbe8ed96147d4b98134c0a95beec9029368b0c35a1190f40094ee186478.d789d64ebfe299b
0e416afc4a169632f903f693095b4629a7ea271d5a0cf2c99
creating metadata file for /root/.cache/huggingface/transformers/571aebbe8ed9614
7d4b98134c0a95beec9029368b0c35a1190f40094ee186478.d789d64ebfe299b0e416afc4a16963
2f903f693095b4629a7ea271d5a0cf2c99
https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/tokenizer.json not found in cache or force_download set to
True, downloading to /root/.cache/huggingface/transformers/tmpd1krimii

Downloading:   0%|          | 0.00/455k [00:00<?, ?B/s]


storing https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/tokenizer.json in cache at /root/.cache/huggingface/transfor
mers/28675318b6e2e081432f0e2fa9539dd176e7765623809f220b03cbafaaf810ad.3d26c56b35
8cca40b16bcdc782f4b906e5c295e21e24fae62803895dae040f25
creating metadata file for /root/.cache/huggingface/transformers/28675318b6e2e08
1432f0e2fa9539dd176e7765623809f220b03cbafaaf810ad.3d26c56b358cca40b16bcdc782f4b9
06e5c295e21e24fae62803895dae040f25
```

```
https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/special_tokens_map.json not found in cache or force_download
set to True, downloading to /root/.cache/huggingface/transformers/tmp31zr80jt

Downloading:   0%|          | 0.00/112 [00:00<?, ?B/s]


storing https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/special_tokens_map.json in cache at /root/.cache/huggingface
/transformers/8059125746dc8b7c4ab020d7078038ce3ba711f8ee2be055be536ac398300fe1.d
d8bd9bfd3664b530ea4e645105f557769387b3da9f79bdb55ed556bdd80611d
creating metadata file for /root/.cache/huggingface/transformers/8059125746dc8b7
c4ab020d7078038ce3ba711f8ee2be055be536ac398300fe1.dd8bd9bfd3664b530ea4e645105f55
7769387b3da9f79bdb55ed556bdd80611d
loading file https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/vocab.txt from cache at /root/.cache/huggingface/transformer
s/571aebbe8ed96147d4b98134c0a95beec9029368b0c35a1190f40094ee186478.d789d64ebfe29
9b0e416afc4a169632f903f693095b4629a7ea271d5a0cf2c99
loading file https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/tokenizer.json from cache at /root/.cache/huggingface/transf
ormers/28675318b6e2e081432f0e2fa9539dd176e7765623809f220b03cbafaaf810ad.3d26c56b
358cca40b16bcdc782f4b906e5c295e21e24fae62803895dae040f25
loading file https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/added_tokens.json from cache at None
loading file https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/special_tokens_map.json from cache at /root/.cache/huggingfa
ce/transformers/8059125746dc8b7c4ab020d7078038ce3ba711f8ee2be055be536ac398300fe1
.dd8bd9bfd3664b530ea4e645105f557769387b3da9f79bdb55ed556bdd80611d
loading file https://huggingface.co/mvonwyl/distilbert-base-uncased-finetuned-
squad2/resolve/main/tokenizer_config.json from cache at /root/.cache/huggingface
/transformers/0eefcbdecb44adc8a8a31203c6fc6081e4abee730e80b3d7c73ae8149d76268e.4
2154c5fd30bfa7e34941d0d8ad26f8a3936990926fbe06b2da76dd749b1c6d4
PyTorch: setting up devices
The default value for the training argument `--report_to` will change in v5
(from all installed integrations to none). In v5, you will need to use
`--report_to all` to get the same behavior as now. You should start updating
your code and make this info disappear :-).

  0%|          | 0/6 [00:00<?, ?ba/s]


  0%|          | 0/6 [00:00<?, ?ba/s]


***** Running training *****
  Num examples = 5273
  Num Epochs = 3
  Instantaneous batch size per device = 8
  Total train batch size (w. parallel, distributed & accumulation) = 8
  Gradient Accumulation steps = 1
  Total optimization steps = 1980

<IPython.core.display.HTML object>


Saving model checkpoint to test_model_prof-finetuned-squad/checkpoint-500
Configuration saved in test_model_prof-finetuned-
squad/checkpoint-500/config.json
Model weights saved in test_model_prof-finetuned-
squad/checkpoint-500/pytorch_model.bin
tokenizer config file saved in test_model_prof-finetuned-
squad/checkpoint-500/tokenizer_config.json
Special tokens file saved in test_model_prof-finetuned-
squad/checkpoint-500/special_tokens_map.json
Saving model checkpoint to test_model_prof-finetuned-squad/checkpoint-1000
Configuration saved in test_model_prof-finetuned-
squad/checkpoint-1000/config.json
Model weights saved in test_model_prof-finetuned-
squad/checkpoint-1000/pytorch_model.bin
tokenizer config file saved in test_model_prof-finetuned-
squad/checkpoint-1000/tokenizer_config.json
Special tokens file saved in test_model_prof-finetuned-
squad/checkpoint-1000/special_tokens_map.json
Saving model checkpoint to test_model_prof-finetuned-squad/checkpoint-1500
Configuration saved in test_model_prof-finetuned-
squad/checkpoint-1500/config.json
Model weights saved in test_model_prof-finetuned-
squad/checkpoint-1500/pytorch_model.bin
tokenizer config file saved in test_model_prof-finetuned-
squad/checkpoint-1500/tokenizer_config.json
Special tokens file saved in test_model_prof-finetuned-
squad/checkpoint-1500/special_tokens_map.json


Training completed. Do not forget to share your model on huggingface.co/models
=)



  0%|          | 0/6 [00:00<?, ?ba/s]


The following columns in the test set  don't have a corresponding argument in
`DistilBertForQuestionAnswering.forward` and have been ignored: offset_mapping,
example_id.
***** Running Prediction *****
  Num examples = 6072
  Batch size = 8
```

```
<IPython.core.display.HTML object>


Post-processing 5937 example predictions split into 6072 features.

  0%|          | 0/5937 [00:00<?, ?it/s]
```

[20]:
```
metric.compute(predictions=formatted_predictions2, references=references2)
```

[20]:
```
{'HasAns_exact': 62.31393775372125,
 'HasAns_f1': 70.89900675793425,
 'HasAns_total': 2956,
 'NoAns_exact': 58.705132505870516,
 'NoAns_f1': 58.705132505870516,
 'NoAns_total': 2981,
 'best_exact': 60.55246757621695,
 'best_exact_thresh': 0.0,
 'best_f1': 64.80778786010966,
 'best_f1_thresh': 0.0,
 'exact': 60.50193700522149,
 'f1': 64.77639615571039,
 'total': 5937}
```

[44]:
```
for i in range(10):
  print("\nPrediction  i =", i)
  print("Expected :", formatted_predictions2[i]["prediction_text"])
  print("Obtained : ", references2[i]["answers"]["text"])
```

```

Prediction  i = 0
Expected : Battle of Bch ng
Obtained :  ['Battle of Bch ng', 'Battle of Bch ng', 'the Battle of Bch
ng']

Prediction  i = 1
Expected :
Obtained :  []

Prediction  i = 2
Expected :
Obtained :  ['Until the oil shock', 'the oil shock', 'the oil shock', 'Until the
oil shock', 'the oil shock']

Prediction  i = 3
Expected : Gaussian primes
Obtained :  ['4k + 3', '4k + 3', 'Z']

Prediction  i = 4
```

```
Expected : Los Angeles central business district
Obtained :  []

Prediction  i = 5
Expected :
Obtained :  []

Prediction  i = 6
Expected : granaries
Obtained :  ['granaries were ordered built throughout the empire', 'granaries
were ordered built throughout the empire', 'granaries were ordered built']

Prediction  i = 7
Expected :
Obtained :  ['gathered in the streets', 'gathered in the streets', 'gathered in
the streets', 'gathered in the streets', 'gathered in the streets']

Prediction  i = 8
Expected :
Obtained :  ['British merchants or fur-traders, Céloron informed them of the
French claims on the territory and told them to leave.', 'told them to leave',
'Whenever he encountered British merchants or fur-traders, Céloron informed them
of the French claims on the territory and told them to leave.', 'buried lead
plates', 'Whenever he encountered British merchants or fur-traders, Céloron
informed them of the French claims on the territory and told them to leave']

Prediction  i = 9
Expected : Saul Bellow
Obtained :  []
```