SRISHTI SHARMA

Hugging Face Transformers is a popular library in the field of Natural Language Processing (NLP) that provides pre-trained models and tools for various NLP tasks, including question answering (QA). The library is built on top of PyTorch and TensorFlow, and it offers an extensive collection of transformer-based models.

```
pip --version
```

```
pip 23.1.2 from /usr/local/lib/python3.10/dist-packages/pip (python 3.10)
```

```
pip install datasets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-packages (2.12.
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: dill<0.3.7,>=0.3.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from da
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from da
Requirement already satisfied: multiprocess in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: fsspec[http]>=2021.11.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from d
Requirement already satisfied: huggingface-hub<1.0.0,>=0.11.0 in /usr/local/lib/python3.1
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: responses<0.19 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: charset-normalizer<4.0,>=2.0 in /usr/local/lib/python3.10/
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/d
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/di
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
```

```
from datasets import load_dataset

raw_datasets = load_dataset("squad")
```

```
WARNING:datasets.builder:Found cached dataset squad (/root/.cache/huggingface/datasets/sq
100%                                                    2/2 [00:00<00:00, 2.26it/s]
```

```
raw_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 87599
    })
    validation: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 10570
    })
})
```

```
print("Context: ", raw_datasets["train"][0]["context"])
print("Question: ", raw_datasets["train"][0]["question"])
print("Answer: ", raw_datasets["train"][0]["answers"])
```

```
Context:  Architecturally, the school has a Catholic character. Atop the Main Building's g
Question:  To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France?
Answer:  {'text': ['Saint Bernadette Soubirous'], 'answer_start': [515]}
```

```
raw_datasets["train"].filter(lambda x: len(x["answers"]["text"]) != 1)
```

```
WARNING:datasets.arrow_dataset:Loading cached processed dataset at /root/.cache/huggingfac
Dataset({
    features: ['id', 'title', 'context', 'question', 'answers'],
    num_rows: 0
})
```

```
print(raw_datasets["validation"][0]["answers"])
print(raw_datasets["validation"][2]["answers"])
```

```
{'text': ['Denver Broncos', 'Denver Broncos', 'Denver Broncos'], 'answer_start': [177, 177
{'text': ['Santa Clara, California', "Levi's Stadium", "Levi's Stadium in the San Francisc
```

```
print(raw_datasets["validation"][2]["context"])
print(raw_datasets["validation"][2]["question"])
```

```
Super Bowl 50 was an American football game to determine the champion of the National Foot
Where did Super Bowl 50 take place?
```

Processing the training data

```
pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: huggingface-hub<1.0,>=0.14.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages 
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from hug
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dis
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (fr
```

```
from transformers import AutoTokenizer

model_checkpoint = "bert-base-cased"
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
```

```
tokenizer.is_fast
```

```
True
```

```
context = raw_datasets["train"][0]["context"]
question = raw_datasets["train"][0]["question"]

inputs = tokenizer(question, context)
tokenizer.decode(inputs["input_ids"])
```

```
'[CLS] To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? [SEP] Arch
itecturally, the school has a Catholic character. Atop the Main Building\'s gold dome is
a golden statue of the Virgin Mary. Immediately in front of the Main Building and facing
it, is a copper statue of Christ with arms upraised with the legend " Venite Ad Me Omnes
". Next to the Main Building is the Basilica of the Sacred Heart. Immediately behind the
basilica is the Grotto, a Marian place of prayer and reflection. It is a replica of the g
rotto at Lourdes, France where the Virgin Mary reputedly appeared to Saint Bernadette Sou
birous in 1858. At the end of the main drive ( and in a direct line that connects through
3 statues and the Gold Dome ), is a simple, modern stone statue of Mary. [SEP]'
```

```
inputs = tokenizer(
    question,
    context,
    max_length=100,
    truncation="only_second",
    stride=50,
    return_overflowing_tokens=True,
)

for ids in inputs["input_ids"]:
    print(tokenizer.decode(ids))
```

```
[CLS] To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? [SEP] Archit
[CLS] To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? [SEP] the Ma
[CLS] To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? [SEP] Next t
[CLS] To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? [SEP]. It is
```

```
inputs = tokenizer(
    question,
    context,
    max_length=100,
    truncation="only_second",
    stride=50,
    return_overflowing_tokens=True,
    return_offsets_mapping=True,
)
inputs.keys()
```

```
dict_keys(['input_ids', 'token_type_ids', 'attention_mask', 'offset_mapping',
'overflow_to_sample_mapping'])
```

```
inputs["overflow_to_sample_mapping"]
```

```
[0, 0, 0, 0]
```

```
inputs = tokenizer(
    raw_datasets["train"][2:6]["question"],
    raw_datasets["train"][2:6]["context"],
    max_length=100,
    truncation="only_second",
    stride=50,
    return_overflowing_tokens=True,
    return_offsets_mapping=True,
)

print(f"The 4 examples gave {len(inputs['input_ids'])} features.")
print(f"Here is where each comes from: {inputs['overflow_to_sample_mapping']}.")
```

```
The 4 examples gave 19 features.
Here is where each comes from: [0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3].
```

```
answers = raw_datasets["train"][2:6]["answers"]
start_positions = []
end_positions = []

for i, offset in enumerate(inputs["offset_mapping"]):
    sample_idx = inputs["overflow_to_sample_mapping"][i]
    answer = answers[sample_idx]
    start_char = answer["answer_start"][0]
    end_char = answer["answer_start"][0] + len(answer["text"][0])
    sequence_ids = inputs.sequence_ids(i)

    # Find the start and end of the context
    idx = 0
    while sequence_ids[idx] != 1:
        idx += 1
    context_start = idx
    while sequence_ids[idx] == 1:
        idx += 1
    context_end = idx - 1

    # If the answer is not fully inside the context, label is (0, 0)
    if offset[context_start][0] > start_char or offset[context_end][1] < end_char:
        start_positions.append(0)
        end_positions.append(0)
    else:
        # Otherwise it's the start and end token positions
        idx = context_start
        while idx <= context_end and offset[idx][0] <= start_char:
            idx += 1
        start_positions.append(idx - 1)

        idx = context_end
        while idx >= context_start and offset[idx][1] >= end_char:
            idx -= 1
        end_positions.append(idx + 1)

start_positions, end_positions
```

```
([83, 51, 19, 0, 0, 64, 27, 0, 34, 0, 0, 0, 67, 34, 0, 0, 0, 0, 0],
 [85, 53, 21, 0, 0, 70, 33, 0, 40, 0, 0, 0, 68, 35, 0, 0, 0, 0, 0])
```

```
idx = 0
sample_idx = inputs["overflow_to_sample_mapping"][idx]
answer = answers[sample_idx]["text"][0]

start = start_positions[idx]
end = end_positions[idx]
labeled_answer = tokenizer.decode(inputs["input_ids"][idx][start : end + 1])

print(f"Theoretical answer: {answer}, labels give: {labeled_answer}")
```

```
Theoretical answer: the Main Building, labels give: the Main Building
```

So that’s a match

```
idx = 4
sample_idx = inputs["overflow_to_sample_mapping"][idx]
answer = answers[sample_idx]["text"][0]

decoded_example = tokenizer.decode(inputs["input_ids"][idx])
print(f"Theoretical answer: {answer}, decoded example: {decoded_example}")
```

```
Theoretical answer: a Marian place of prayer and reflection, decoded example: [CLS] What i
```

we don’t see the answer inside the context

```
max_length = 384
stride = 128


def preprocess_training_examples(examples):
    questions = [q.strip() for q in examples["question"]]
    inputs = tokenizer(
        questions,
        examples["context"],
        max_length=max_length,
        truncation="only_second",
        stride=stride,
        return_overflowing_tokens=True,
        return_offsets_mapping=True,
        padding="max_length",
    )

    offset_mapping = inputs.pop("offset_mapping")
    sample_map = inputs.pop("overflow_to_sample_mapping")
    answers = examples["answers"]
    start_positions = []
    end_positions = []

    for i, offset in enumerate(offset_mapping):
        sample_idx = sample_map[i]
        answer = answers[sample_idx]
        start_char = answer["answer_start"][0]
        end_char = answer["answer_start"][0] + len(answer["text"][0])
        sequence_ids = inputs.sequence_ids(i)

        # Find the start and end of the context
        idx = 0
        while sequence_ids[idx] != 1:
            idx += 1
        context_start = idx
        while sequence_ids[idx] == 1:
            idx += 1
        context_end = idx - 1

        # If the answer is not fully inside the context, label is (0, 0)
        if offset[context_start][0] > start_char or offset[context_end][1] < end_char:
            start_positions.append(0)
            end_positions.append(0)
        else:
            # Otherwise it's the start and end token positions
            idx = context_start
            while idx <= context_end and offset[idx][0] <= start_char:
                idx += 1
            start_positions.append(idx - 1)

            idx = context_end
            while idx >= context_start and offset[idx][1] >= end_char:
                idx -= 1
            end_positions.append(idx + 1)

    inputs["start_positions"] = start_positions
    inputs["end_positions"] = end_positions
    return inputs
```

```
train_dataset = raw_datasets["train"].map(
    preprocess_training_examples,
    batched=True,
    remove_columns=raw_datasets["train"].column_names,)
len(raw_datasets["train"]), len(train_dataset)
```

```
WARNING:datasets.arrow_dataset:Loading cached processed dataset at /root/.cache/huggingfac
(87599, 88729)
```

the preprocessing added roughly 1,000 features

Processing the validation data

```
def preprocess_validation_examples(examples):
    questions = [q.strip() for q in examples["question"]]
    inputs = tokenizer(
        questions,
        examples["context"],
        max_length=max_length,
        truncation="only_second",
        stride=stride,
        return_overflowing_tokens=True,
        return_offsets_mapping=True,
        padding="max_length",
    )

    sample_map = inputs.pop("overflow_to_sample_mapping")
    example_ids = []

    for i in range(len(inputs["input_ids"])):
        sample_idx = sample_map[i]
        example_ids.append(examples["id"][sample_idx])

        sequence_ids = inputs.sequence_ids(i)
        offset = inputs["offset_mapping"][i]
        inputs["offset_mapping"][i] = [
            o if sequence_ids[k] == 1 else None for k, o in enumerate(offset)
        ]

    inputs["example_id"] = example_ids
    return inputs
```

```
validation_dataset = raw_datasets["validation"].map(
    preprocess_validation_examples,
    batched=True,
    remove_columns=raw_datasets["validation"].column_names,
)
len(raw_datasets["validation"]), len(validation_dataset)
```

```
(10570, 10822)
```

In this case we've only added a couple of hundred samples, so it appears the contexts in the validation dataset are a bit shorter.

Fine-tuning the model with the Trainer API

Post Processing

```
small_eval_set = raw_datasets["validation"].select(range(100))
trained_checkpoint = "distilbert-base-cased-distilled-squad"

tokenizer = AutoTokenizer.from_pretrained(trained_checkpoint)
eval_set = small_eval_set.map(
    preprocess_validation_examples,
    batched=True,
    remove_columns=raw_datasets["validation"].column_names,
)
```

```
WARNING:datasets.arrow_dataset:Loading cached processed dataset at /root/.cache/huggingfac
```

```
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint)
```

```
import torch
from transformers import AutoModelForQuestionAnswering

eval_set_for_model = eval_set.remove_columns(["example_id", "offset_mapping"])
eval_set_for_model.set_format("torch")

device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
batch = {k: eval_set_for_model[k].to(device) for k in eval_set_for_model.column_names}
trained_model = AutoModelForQuestionAnswering.from_pretrained(trained_checkpoint).to(
    device)

with torch.no_grad():
    outputs = trained_model(**batch)
```

Since the Trainer will give us predictions as NumPy arrays, we grab the start and end logits and convert them to that format

```
start_logits = outputs.start_logits.cpu().numpy()
end_logits = outputs.end_logits.cpu().numpy()
```

```
import collections

example_to_features = collections.defaultdict(list)
for idx, feature in enumerate(eval_set):
    example_to_features[feature["example_id"]].append(idx)
```

```
import numpy as np

n_best = 20
max_answer_length = 30
predicted_answers = []

for example in small_eval_set:
    example_id = example["id"]
    context = example["context"]
    answers = []

    for feature_index in example_to_features[example_id]:
        start_logit = start_logits[feature_index]
        end_logit = end_logits[feature_index]
        offsets = eval_set["offset_mapping"][feature_index]

        start_indexes = np.argsort(start_logit)[-1 : -n_best - 1 : -1].tolist()
        end_indexes = np.argsort(end_logit)[-1 : -n_best - 1 : -1].tolist()
        for start_index in start_indexes:
            for end_index in end_indexes:
                # Skip answers that are not fully in the context
                if offsets[start_index] is None or offsets[end_index] is None:
                    continue
                # Skip 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

                answers.append(
                    {
                        "text": context[offsets[start_index][0] : offsets[end_index][1]],
                        "logit_score": start_logit[start_index] + end_logit[end_index],
                }
            )

    best_answer = max(answers, key=lambda x: x["logit_score"])
    predicted_answers.append({"id": example_id, "prediction_text": best_answer["text"]})
```

```
pip install evaluate
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
Requirement already satisfied: evaluate in /usr/local/lib/python3.10/dist-packages (0.4.0)
Requirement already satisfied: datasets>=2.0.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: dill in /usr/local/lib/python3.10/dist-packages (from evalu
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from eva
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from eva
Requirement already satisfied: multiprocess in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: fsspec[http]>=2021.05.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: huggingface-hub>=0.7.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: responses<0.19 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from da
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from h
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dis
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.10/di
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from p
```

```
import evaluate

metric = evaluate.load("squad")
```

```
theoretical_answers = [
    {"id": ex["id"], "answers": ex["answers"]} for ex in small_eval_set
]
```

```
print(predicted_answers[0])
print(theoretical_answers[0])
```

```
{'id': '56be4db0acb8001400a502ec', 'prediction_text': 'Denver Broncos'}
{'id': '56be4db0acb8001400a502ec', 'answers': {'text': ['Denver Broncos', 'Denver Broncos
```

```
metric.compute(predictions=predicted_answers, references=theoretical_answers)
```

```
{'exact_match': 83.0, 'f1': 88.25000000000004}
```

Now let’s put everything we just did in a compute_metrics() function that we will use in the Trainer

```
from tqdm.auto import tqdm


def compute_metrics(start_logits, end_logits, features, examples):
    example_to_features = collections.defaultdict(list)
    for idx, feature in enumerate(features):
        example_to_features[feature["example_id"]].append(idx)

    predicted_answers = []
    for example in tqdm(examples):
        example_id = example["id"]
        context = example["context"]
        answers = []

        # Loop through all features associated with that example
        for feature_index in example_to_features[example_id]:
            start_logit = start_logits[feature_index]
            end_logit = end_logits[feature_index]
            offsets = features[feature_index]["offset_mapping"]

            start_indexes = np.argsort(start_logit)[-1 : -n_best - 1 : -1].tolist()
            end_indexes = np.argsort(end_logit)[-1 : -n_best - 1 : -1].tolist()
            for start_index in start_indexes:
                for end_index in end_indexes:
                    # Skip answers that are not fully in the context
                    if offsets[start_index] is None or offsets[end_index] is None:
                        continue
                    # Skip 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

                    answer = {
                        "text": context[offsets[start_index][0] : offsets[end_index][1]],
                        "logit_score": start_logit[start_index] + end_logit[end_index],
                    }
                    answers.append(answer)

        # Select the answer with the best score
        if len(answers) > 0:
            best_answer = max(answers, key=lambda x: x["logit_score"])
            predicted_answers.append(
                {"id": example_id, "prediction_text": best_answer["text"]}
            )
        else:
            predicted_answers.append({"id": example_id, "prediction_text": ""})

    theoretical_answers = [{"id": ex["id"], "answers": ex["answers"]} for ex in examples]
    return metric.compute(predictions=predicted_answers, references=theoretical_answers)
```

```
compute_metrics(start_logits, end_logits, eval_set, small_eval_set)
```

```
100%                                                   100/100 [00:00<00:00, 619.69it/s]
```

Fine-tuning the model

```
model = AutoModelForQuestionAnswering.from_pretrained(model_checkpoint)
```

```
Some weights of the model checkpoint at bert-base-cased were not used when initializing Be
- This IS expected if you are initializing BertForQuestionAnswering from the checkpoint of
- This IS NOT expected if you are initializing BertForQuestionAnswering from the checkpoir
Some weights of BertForQuestionAnswering were not initialized from the model checkpoint at
You should probably TRAIN this model on a down-stream task to be able to use it for predic
```

```
from huggingface_hub import notebook_login

notebook_login()
```

```
               Token is valid (permission: write).

‹en has been saved in your configured git credential helpers

/our token has been saved to /root/.cache/huggingface/toker

                     Login successful
```

```
pip install --upgrade accelerate
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
Requirement already satisfied: accelerate in /usr/local/lib/python3.10/dist-packages (0.19
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from acc
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from acc
Requirement already satisfied: torch>=1.6.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torc
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from tor
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from trit
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from tritor
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (fr
```

```
pip install git+https://github.com/huggingface/accelerate
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
Collecting git+https://github.com/huggingface/accelerate
  Cloning https://github.com/huggingface/accelerate to /tmp/pip-req-build-v6op_4ny
  Running command git clone --filter=blob:none --quiet https://github.com/huggingface/acce
  Resolved https://github.com/huggingface/accelerate to commit 85901cdcf99e9fd258811789ca6
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages 
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from acc
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from acc
Requirement already satisfied: torch>=1.6.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torc
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from tor
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from trit
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from trito
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages 
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (fr
Building wheels for collected packages: accelerate
  Building wheel for accelerate (pyproject.toml) ... done
  Created wheel for accelerate: filename=accelerate-0.20.0.dev0-py3-none-any.whl size=2269
  Stored in directory: /tmp/pip-ephem-wheel-cache-na6dr0au/wheels/f6/c7/9d/1b8a5ca8353d936
Successfully built accelerate
Installing collected packages: accelerate
  Attempting uninstall: accelerate
    Found existing installation: accelerate 0.19.0
    Uninstalling accelerate-0.19.0:
      Successfully uninstalled accelerate-0.19.0
```

```
pip uninstall -y transformers accelerate
```

```
Found existing installation: transformers 4.28.0
Uninstalling transformers-4.28.0:
  Successfully uninstalled transformers-4.28.0
Found existing installation: accelerate 0.20.0.dev0
Uninstalling accelerate-0.20.0.dev0:
  Successfully uninstalled accelerate-0.20.0.dev0
```

```
pip install transformers accelerate
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public
Collecting transformers
  Using cached transformers-4.29.2-py3-none-any.whl (7.1 MB)
Collecting accelerate
  Using cached accelerate-0.19.0-py3-none-any.whl (219 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: huggingface-hub<1.0,>=0.14.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from acc
Requirement already satisfied: torch>=1.6.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from hug
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dis
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torc
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from t
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from tor
```

```python
from transformers import TrainingArguments

args = TrainingArguments(
    "bert-finetuned-squad",
    evaluation_strategy="no",
    save_strategy="epoch",
    learning_rate=2e-5,
    num_train_epochs=3,
    weight_decay=0.01,
    # fp16=True,
    push_to_hub=True)
```

```
  [accelerate,transformers]
```

```python
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=validation_dataset,
    tokenizer=tokenizer,)
trainer.train()
```

```
Cloning https://huggingface.co/srish884/bert-finetuned-squad into local empty directory.
WARNING:huggingface_hub.repository:Cloning https://huggingface.co/srish884/bert-finetuned-
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:407: FutureWarning: T
  warnings.warn(
You're using a BertTokenizerFast tokenizer. Please note that with a fast tokenizer, using
```

[124/33276 50:03 < 226:43:56, 0.04 it/s, Epoch 0.01/3]

Step	Training Loss

[235/33276 1:36:22 < 227:46:28, 0.04 it/s, Epoch 0.02/3]

Step	Training Loss

The whole Training takes over an hour

```
# predictions, _, _ = trainer.predict(validation_dataset)
# start_logits, end_logits = predictions
# compute_metrics(start_logits, end_logits, validation_dataset, raw_datasets["validation"])
```

```
# trainer.push_to_hub(commit_message="Training complete")
```

```
# from torch.utils.data import DataLoader
# from transformers import default_data_collator

# train_dataset.set_format("torch")
# validation_set = validation_dataset.remove_columns(["example_id", "offset_mapping"])
# validation_set.set_format("torch")

# train_dataloader = DataLoader(
#     train_dataset,
#     shuffle=True,
#     collate_fn=default_data_collator,
#     batch_size=8,
# )
# eval_dataloader = DataLoader(
#     validation_set, collate_fn=default_data_collator, batch_size=8
)
```

```
# model = AutoModelForQuestionAnswering.from_pretrained(model_checkpoint)
```

```
# from torch.optim import AdamW

# optimizer = AdamW(model.parameters(), lr=2e-5)
```

```
# from accelerate import Accelerator

# accelerator = Accelerator(fp16=True)
# model, optimizer, train_dataloader, eval_dataloader = accelerator.prepare(
#     model, optimizer, train_dataloader, eval_dataloader
)
```

```
# from transformers import get_scheduler

# num_train_epochs = 3
# num_update_steps_per_epoch = len(train_dataloader)
# num_training_steps = num_train_epochs * num_update_steps_per_epoch

# lr_scheduler = get_scheduler(
#     "linear",
#     optimizer=optimizer,
#     num_warmup_steps=0,
#     num_training_steps=num_training_steps,
)
```

```
# from transformers import get_scheduler

# num_train_epochs = 3
# num_update_steps_per_epoch = len(train_dataloader)
# num_training_steps = num_train_epochs * num_update_steps_per_epoch

# lr_scheduler = get_scheduler(
#     "linear",
#     optimizer=optimizer,
#     num_warmup_steps=0,
#     num_training_steps=num_training_steps)
```

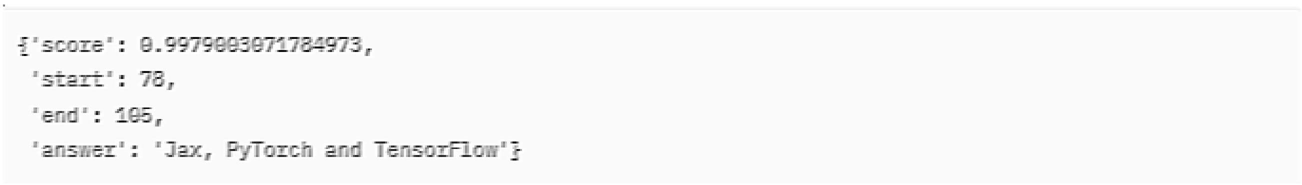
```
# output_dir = "bert-finetuned-squad-accelerate"
# repo = Repository(output_dir, clone_from=repo_name)
```

The complete code for the training loop

```
# from tqdm.auto import tqdm
# import torch

# progress_bar = tqdm(range(num_training_steps))

# for epoch in range(num_train_epochs):
#     # Training
#     model.train()
#     for step, batch in enumerate(train_dataloader):
#         outputs = model(**batch)
#         loss = outputs.loss
#         accelerator.backward(loss)

#         optimizer.step()
#         lr_scheduler.step()
#         optimizer.zero_grad()
#         progress_bar.update(1)

#     # Evaluation
#     model.eval()
#     start_logits = []
#     end_logits = []
#     accelerator.print("Evaluation!")
#     for batch in tqdm(eval_dataloader):
#         with torch.no_grad():
#             outputs = model(**batch)

#         start_logits.append(accelerator.gather(outputs.start_logits).cpu().numpy())
#         end_logits.append(accelerator.gather(outputs.end_logits).cpu().numpy())

#     start_logits = np.concatenate(start_logits)
#     end_logits = np.concatenate(end_logits)
#     start_logits = start_logits[: len(validation_dataset)]
#     end_logits = end_logits[: len(validation_dataset)]

#     metrics = compute_metrics(
#         start_logits, end_logits, validation_dataset, raw_datasets["validation"]
#     )
#     print(f"epoch {epoch}:", metrics)

#     # Save and upload
#     accelerator.wait_for_everyone()
#     unwrapped_model = accelerator.unwrap_model(model)
#     unwrapped_model.save_pretrained(output_dir, save_function=accelerator.save)
#     if accelerator.is_main_process:
#         tokenizer.save_pretrained(output_dir)
#         repo.push_to_hub(
#             commit_message=f"Training in progress epoch {epoch}", blocking=False
#         )
```

```
# accelerator.wait_for_everyone()
# unwrapped_model = accelerator.unwrap_model(model)
# unwrapped_model.save_pretrained(output_dir, save_function=accelerator.save)
```

```
from transformers import pipeline

model_checkpoint = "https://huggingface.co/srish884/bert-finetuned-squad"
question_answerer = pipeline("question-answering", model=model_checkpoint)

context = """
HuggingFace Transformers is backed by the three most popular deep learning libraries — Jax, PyTorch an
between them. It's straightforward to train your models with one before loading them for inference wit
"""
question = "Which deep learning libraries back HuggingFace Transformers?"
question_answerer(question=question, context=context)
```



```
{'score': 0.9979003071784973,
 'start': 78,
 'end': 105,
 'answer': 'Jax, PyTorch and TensorFlow'}
```

27m 27s completed at 10:07 PM