Using BERT-like models for extractive reading comprehension on the SQuAD2.0

ELEC-E5550 Statistical Natural Language Processing

28.4.2022

Joona Rantanen

Pablo Carabana Garcia

Saku Komulainen

Introduction

Extractive Reading Comprehension (RC) systems are models where they are given a document and a question about the document. The objective of the model is to find and identify the answer to the question found in the document. The extractive RC model should only use the given document to answer the question without using previous knowledge about the topic (Asai. et al., 2018). The most prevalent dataset for assessing the performance of the models is the Stanford Question Answering Dataset discussed in the next section. The problem of extractive reading comprehension is typically approached with deep learning models.

The Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset which consists of questions posed by crowdworkers on a set of wikipedia articles. In total it contains over 107,785 question-answer pairs on 536 articles. The data contains answers to each question as a span or segment in the corresponding reading passage. (Rajpurkar, et al. 2016) The SQuAD2.0 is the latest version and it combines 100,000 questions from SQuAD1.1 with over 50,000 unanswered questions which are written adversially by crowdworkers (SQuAD2.0 n.d). The performance of the models working with the SQuAD dataset are measured with Exact match (EM), which refers to the percentage of exactly true answers, and with F1-score. The F1-score is a harmonic mean of the precision and recall of the predictions. Since 2018, the human performance (86.831 EM and 89.452 F1-score) on the SQuAD2.0 dataset has been surpassed by multiple machine learning models (SQuAD2.0, n.d).

Bidirectional Encoder Representations from Transformers (BERT) is a language model introduced by Google in 2018 (Devlin, et al. 2018). BERT is based on the Transformer (Vaswani, et al. 2017) architecture introduced in 2017. The Transformer uses a self-attention mechanism which allows the input sequence to attend to itself in every position in the sequence. By doing this, it achieved many state of the art results in various NLP tasks and therefore it became the foundation for many language models including BERT.

In this work we fine-tuned and compared different BERT models on the SQuAD2.0 dataset. Using multiple BERT models allows us to learn the best compromise between computational time, required resources and question answering performance. For example, how does BERT-tiny compare to BERT-small or BERT-base. This is an interesting question, as the larger models require more VRAM on the GPU and in general require more resources, which make large BERT models infeasible on lower powered devices, such as IoT devices.

Methods

In this project we compared six different bert models on the question answering task. The models used in this work are BERT-Base, BERT-Small, BERT-Tiny, DistilBERT, ALBERT and RoBERTa. All the models are finetuned with the SQuAD2.0 dataset.

The project was implemented with mainly TensorFlow and Hugging Face libraries. The tensorflow was used to train and interpret the models and Hugging Face to obtain the pre-trained BERT models.

The BERT-small and BERT-Tiny models were downloaded from https://github.com/google-research/bert, which were converted to a suitable form for HuggingFace transformers library. The other models were obtained directly from the HuggingFace models library.

This project uses some code from the Hugging Faces question answering notebook, which can be found from https://colab.research.google.com/github/huggingface/notebooks/blob/main/examples/question_answering-tf.ipynb. Most importantly we took the preprocessing functions from this notebook, modified them and used them in our project.

Experiments

The finetuning of the BERT like models were run on a computer with Intel i7-10700k, 32GB ram and Nvidia RTX 3070Ti(8GB Vram). Majority of the computations, including the fitting/finetuning and predictions were done on the RTX 3070Ti. We used a batch size of four and three epochs to train each of these models. The models were first trained for two epochs and later the training was continued for one more epoch. The third epoch of training did not significantly improve the training or validation loss for any of the models. The optimizer used was Adam with initial learning rate 2e-5.

Results

The training and validation losses were obtained from the training process, but the main interest of our project are the EM and F1-scores. Additionally, the sizes of the resulting models were noted. The result of the six different bert like models are summarized in the table below:

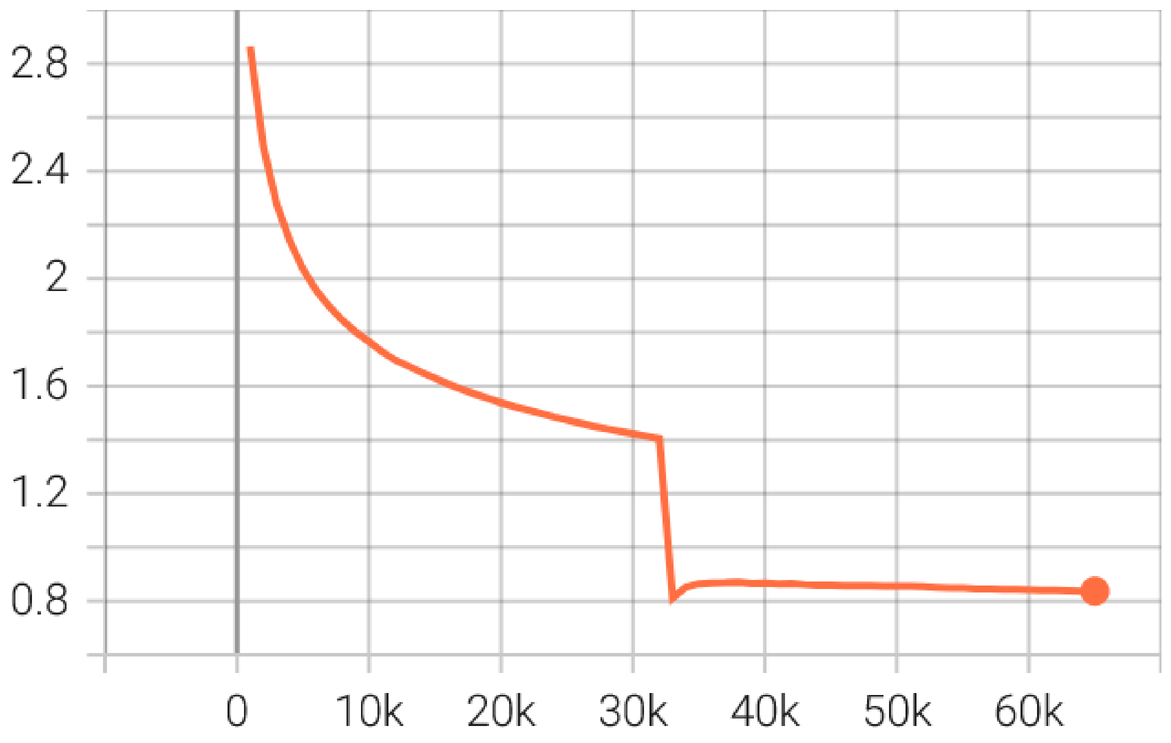
	training loss	validation loss	exact match (EM)	F1-score	Model size (megabytes)
BERT-Base	**0.67**	**1.20**	**71.41**	**74.39**	415MB
BERT-Small	1.08	1.36	62.74	66.06	108MB
BERT-Tiny	2.46	2.05	47.81	49.21	17MB
DistilBERT	0.78	1.25	67.83	71.37	253MB
ALBERT	1.59	1.59	54.97	58.96	42MB
RoBERTa	1.49	1.41	60.98	64.96	473MB

The values look reasonable for all of the models in the table above, however, they do not show the whole picture. The table below contains the EM and F1-scores separately for questions which do or do not have answers:

	HasAns EM	HasAns F1	NoAns EM	NoAns F1
BERT-Base	**71.58**	**77.54**	71.25	71.25
BERT-Small	60.39	67.05	65.08	65.08
BERT-Tiny	15.11	17.90	**80.42**	**80.42**
DistilBERT	66.90	73.98	68.76	68.76
ALBERT	38.04	46.04	71.84	71.84
RoBERTa	46.69	54.66	75.22	75.22

The table reveals an interesting issue with the EM and F1-score. While the EM for BERT-Tiny is not great at 47.81, it is still reasonable for such a small model. However, the EM and F1-score are only 15.11 and 17.90 respectively for the questions which have an answer. What this tells us is that the BERT-Tiny model is not suitable for the SQuAD 2.0 as is. Similarly, the ALBERT model does not seem to be a good match for the SQuAD 2.0 solely looking at the scores.

We also used TensorFlows TensorBoard to track the training procedure, but resuming the training for more epochs failed to continue graphing properly. As such, the graphs for the first two epochs are shown. All of the models follow the same curve shape for the loss and thus only the graph for DistilBERT is shown.



All of the models converge quite quickly during the second epoch and the following epochs do not significantly improve the losses.

The approximate time taken on the hardware described in the experiments section for one epoch of fine tuning is shown in the table below:

	BERT-Base	BERT-Small	BERT-Tiny	Distil-BERT	ALBERT	RoBERTa
Time(s)	7600	2000	860	4000	6500	7600

While the time taken for fine tuning is an important factor when selecting the models, it should be noted that the fine tuning should only need to be done once. As such, when considering models for resource limited devices, it should not be weighted as heavily as something like the model size and the accuracy of the model.

Conclusions

Overall, the two models that performed the best were BERT-Base and DistilBERT. The BERT-Base has a slightly better prediction accuracy, but the required size for the model is almost double of DistilBERT. This makes DistilBERT very attractive for more resource limited devices, while also having acceptable accuracy.

Comparing our results to the results on the SQuAD 2.0 leaderboard, it can be seen that we had some issues with the models' fine tuning. For example, RoBERTa on the SQuAD leaderboard achieved 86.82 exact match and 89.80 F1 -scores, whereas our model achieved only 60.98 and 64.96 respectively. This may stem from using the same tokenizer for all of the models, but we ran out of time before we could try to fine tune the models with their own tokenizers. However, the tokenizer should not have a significant difference on the prediction accuracy, as all of the models are based on BERT.

At a model size of 253MB DistilBERT may still be feasible for use in IoT applications, however it is starting to push the limits on lower powered devices. For such cases, ALBERT with ensemble learning may be a more feasible solution to provide somewhat accurate results. However, using ALBERT as it does not provide good enough question answering accuracy to be useful. But as discussed above, there were some issues with the accuracy of the fine tuned models. On the SQuAD 2.0 leaderboard, ALBERT has achieved 88.59 EM and 91.29 F1-score. With the ALBERT model at only 42MB, it is very attractive for devices with very limited resources, such as IoT devices. For applications where the resources are not as limited, BERT-Base provides the best accuracy from the models we experimented on.

It should be noted that the results on the SQuAD 2.0 leaderboard are obtained from another dataset that is not publicly available, and as such the results are not directly comparable. However, the results from the Dev Set (validation dataset used for evaluation) still give an indication of the performance of the models.

Division of labor

Since we implemented our project in a collaborative notebook environment called deepnote, we were able to program the code together. So everyone did research on the topic and then we together wrote the program code. We also wrote the report together.

Acknowledgments

We did not get any outside help on the project implementation.

References

Asai, A., Eriguchi, A., Hashimoto, K. and Tsuruoka, Y., 2018. Multilingual extractive reading comprehension by runtime machine translation. *arXiv preprint arXiv:1809.03275*. https://arxiv.org/pdf/1809.03275.pdf

Rajpurkar, P., Zhang, J., Lopyrev, K. and Liang, P., 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

SQuAD2.0 n.d., accessed 28 April 2022, <rajpurkar.github.io/SQuAD-explorer>

Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. *Advances in neural information processing systems*, *30*.

Appendix

This appendix contains the program code of the project implementation.

Training code

```
import numpy as np
import tensorflow as tf
print(tf.__version__)
from transformers import AutoTokenizer, TFBertForQuestionAnswering,
DefaultDataCollator, create_optimizer, TFAutoModelForQuestionAnswering
from datasets import load_dataset
from tensorflow.keras.callbacks import TensorBoard
from datetime import datetime

## Load and prepare the data

tokenizer = AutoTokenizer.from_pretrained("./tokenizer")
#.from_pretrained("bert-base-uncased")
max_length = 384
doc_stride = 128
dataset = load_dataset("squad_v2")

# From huggingface.co example
def preprocess_function(examples):
       pad_on_right = tokenizer.padding_side == "right"

       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",
        )

        sample_mapping = tokenized_examples.pop("overflow_to_sample_mapping")
        offset_mapping = tokenized_examples.pop("offset_mapping")

        tokenized_examples["start_positions"] = []
        tokenized_examples["end_positions"] = []

        for i, offsets in enumerate(offset_mapping):
        input_ids = tokenized_examples["input_ids"][i]
        cls_index = input_ids.index(tokenizer.cls_token_id)

        sequence_ids = tokenized_examples.sequence_ids(i)

        sample_index = sample_mapping[i]
        answers = examples["answers"][sample_index]
        if len(answers["answer_start"]) == 0:
                tokenized_examples["start_positions"].append(cls_index)
                tokenized_examples["end_positions"].append(cls_index)
        else:
                start_char = answers["answer_start"][0]
                end_char = start_char + len(answers["text"][0])
                token_start_index = 0
                while sequence_ids[token_start_index] != (1 if pad_on_right else
0):
                token_start_index += 1

                token_end_index = len(input_ids) - 1
                while sequence_ids[token_end_index] != (1 if pad_on_right else 0):
                token_end_index -= 1

                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:
                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
data_tokenized = dataset.map(preprocess_function, batched=True,
remove_columns=dataset["train"].column_names)
print(data_tokenized)
data_collator = DefaultDataCollator(return_tensors="tf")
batch_size = 4

tf_train_set = data_tokenized["train"].to_tf_dataset(
        columns=["attention_mask", "input_ids", "start_positions",
"end_positions"],
        shuffle=True,
        batch_size=batch_size,
        collate_fn=data_collator,
)

tf_validation_set = data_tokenized["validation"].to_tf_dataset(
        columns=["attention_mask", "input_ids", "start_positions",
"end_positions"],
        shuffle=False,
        batch_size=batch_size,
        collate_fn=data_collator,
)
tokenizer.save_pretrained("tokenizer")

## Load and train the models

#### BERT_base

num_epochs = 2
total_train_steps = (len(data_tokenized["train"]) // batch_size) * num_epochs
optimizer, schedule = create_optimizer(
        init_lr=2e-5,
        num_warmup_steps=0,
        num_train_steps=total_train_steps,
)
bert_base = TFBertForQuestionAnswering.from_pretrained("bert-base-uncased")

bert_base.compile(optimizer=optimizer)
tensorboard_callback = TensorBoard(log_dir="./logs/bert_base", update_freq=1000)

bert_base.fit(
        x=tf_train_set,
        validation_data=tf_validation_set,
        epochs=num_epochs,
        callbacks=[tensorboard_callback]
)
```

```python
bert_base.save_pretrained("./finetuned/bert_base")

#### BERT_small

num_epochs = 2
total_train_steps = (len(data_tokenized["train"]) // batch_size) * num_epochs
optimizer, schedule = create_optimizer(
       init_lr=2e-5,
       num_warmup_steps=0,
       num_train_steps=total_train_steps,
)
bert_small =
TFBertForQuestionAnswering.from_pretrained("./bert_small/model.bin",
from_pt=True, config="./bert_small/config.json")
bert_small.compile(optimizer=optimizer)
tensorboard_callback = TensorBoard(log_dir="./logs/bert_small",
update_freq=1000)

bert_small.fit(
       x=tf_train_set,
       validation_data=tf_validation_set,
       epochs=num_epochs,
       callbacks=[tensorboard_callback]
)

bert_small.save_pretrained("./finetuned/bert_small")

#### BERT_tiny

num_epochs = 2
total_train_steps = (len(data_tokenized["train"]) // batch_size) * num_epochs
optimizer, schedule = create_optimizer(
       init_lr=2e-5,
       num_warmup_steps=0,
       num_train_steps=total_train_steps,
)
bert_tiny = TFBertForQuestionAnswering.from_pretrained("./bert_tiny/model.bin",
from_pt=True, config="./bert_tiny/config.json")
bert_tiny.compile(optimizer=optimizer)
tensorboard_callback = TensorBoard(log_dir="./logs/bert_tiny", update_freq=1000)

bert_tiny.fit(
       x=tf_train_set,
       validation_data=tf_validation_set,
       epochs=num_epochs,
       callbacks=[tensorboard_callback]
)

bert_tiny.save_pretrained("./finetuned/bert_tiny")
```

```
#### DistilBERT

num_epochs = 2
total_train_steps = (len(data_tokenized["train"]) // batch_size) * num_epochs
optimizer, schedule = create_optimizer(
	init_lr=2e-5,
	num_warmup_steps=0,
	num_train_steps=total_train_steps,
)
distil_bert =
TFAutoModelForQuestionAnswering.from_pretrained("distilbert-base-uncased")
distil_bert.compile(optimizer=optimizer)
tensorboard_callback = TensorBoard(log_dir="./logs/distil_bert",
update_freq=1000)

distil_bert.fit(
	x=tf_train_set,
	validation_data=tf_validation_set,
	epochs=num_epochs,
	callbacks=[tensorboard_callback]
)

distil_bert.save_pretrained("./finetuned/distil_bert")

#### ALBERT

num_epochs = 2
total_train_steps = (len(data_tokenized["train"]) // batch_size) * num_epochs
optimizer, schedule = create_optimizer(
	init_lr=2e-5,
	num_warmup_steps=0,
	num_train_steps=total_train_steps,
)
albert = TFAutoModelForQuestionAnswering.from_pretrained("albert-base-v2")
albert.compile(optimizer=optimizer)
tensorboard_callback = TensorBoard(log_dir="./logs/albert", update_freq=1000)

albert.fit(
	x=tf_train_set,
	validation_data=tf_validation_set,
	epochs=num_epochs,
	callbacks=[tensorboard_callback]
)

albert.save_pretrained("./finetuned/albert")

#### RoBERTa

num_epochs = 2
total_train_steps = (len(data_tokenized["train"]) // batch_size) * num_epochs
optimizer, schedule = create_optimizer(
        init_lr=2e-5,
        num_warmup_steps=0,
        num_train_steps=total_train_steps,
)
roberta = TFAutoModelForQuestionAnswering.from_pretrained("roberta-base")
roberta.compile(optimizer=optimizer)
tensorboard_callback = TensorBoard(log_dir="./logs/roberta", update_freq=1000)

roberta.fit(
        x=tf_train_set,
        validation_data=tf_validation_set,
        epochs=num_epochs,
        callbacks=[tensorboard_callback]
)

roberta.save_pretrained("./finetuned/roberta")

## Additional training for the models

model_params = [
        "bert_base" ,
        "distil_bert",
        "albert",
        "roberta",
        "bert_small",
        "bert_tiny",
]
num_epochs = 1
total_train_steps = (len(data_tokenized["train"]) // batch_size) * num_epochs

for model_path in model_params:
        print(f"Start training {model_path} at {datetime.now().ctime()}")
        model =
TFAutoModelForQuestionAnswering.from_pretrained(f"./finetuned/{model_path}")

        optimizer, schedule = create_optimizer(
        init_lr=2e-5,
        num_warmup_steps=0,
        num_train_steps=total_train_steps,
        )

        model.compile(optimizer=optimizer)

        tensorboard_callback = TensorBoard(log_dir=f"./logs/{model_path}",
update_freq=1000)
        model.fit(
        x=tf_train_set,
        validation_data=tf_validation_set,
        epochs=num_epochs,
        callbacks=[tensorboard_callback]
        )
```

```
model.save_pretrained(f"./finetuned2/{model_path}")
```

Evaluation code

```
import tensorflow as tf
import pandas as pd
import numpy as np
from transformers import AutoTokenizer, TFAutoModelForQuestionAnswering,
DefaultDataCollator
from datasets import load_dataset, load_metric
import collections
tokenizer = AutoTokenizer.from_pretrained("./tokenizer")
dataset = load_dataset("squad_v2")["validation"]
max_length = 384
doc_stride = 128

# From huggingface.co example
def preprocess_function(examples):
	pad_on_right = tokenizer.padding_side == "right"

	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",
	)

	sample_mapping = tokenized_examples.pop("overflow_to_sample_mapping")

	tokenized_examples["example_id"] = []

	for i in range(len(tokenized_examples["input_ids"])):
	sequence_ids = tokenized_examples.sequence_ids(i)
	context_index = 1 if pad_on_right else 0

	sample_index = sample_mapping[i]
	tokenized_examples["example_id"].append(examples["id"][sample_index])

	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

# From huggingface.co example with some modifications made to work with our data
def postprocess_function(examples, features, all_start_logits, all_end_logits,
n_best_size=20, max_answer_length=30):
    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)

    predictions = collections.OrderedDict()

    for example_index, example in enumerate(examples):
        feature_indices = features_per_example[example_index]

        min_null_score = None
        valid_answers = []

        context = example["context"]
        for feature_index in feature_indices:
            start_logits = all_start_logits[feature_index]
            end_logits = all_end_logits[feature_index]
            offset_mapping = features[feature_index]["offset_mapping"]

            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

            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:
                    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
                        or len(offset_mapping[start_index]) == 0
                        or len(offset_mapping[end_index]) == 0
                    ):
                        continue
                    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:
            best_answer = {"text": "", "score": 0.0}

        answer = (
            best_answer["text"] if best_answer["score"] > min_null_score else ""
        )
        predictions[example["id"]] = answer

    return predictions

data_tokenized = dataset.map(preprocess_function, batched=True,
remove_columns=dataset.column_names)
data_collator = DefaultDataCollator(return_tensors="tf")
batch_size = 4

tf_validation_set = data_tokenized.to_tf_dataset(
      columns=["attention_mask", "input_ids"],
      shuffle=False,
      batch_size=batch_size,
      collate_fn=data_collator
)
## Predict the data using the models
model_paths = [
      "./finetuned2/bert_base",
      "./finetuned2/bert_small",
      "./finetuned2/bert_tiny",
      "./finetuned2/distil_bert",
      "./finetuned2/albert",
      "./finetuned2/roberta"
]

metric = load_metric("squad_v2")
model_metrics = collections.OrderedDict()
```

```
for model_path in model_paths:
       print(model_path)
       model = TFAutoModelForQuestionAnswering.from_pretrained(model_path)
       print("model loaded, predicting...")
       pred = model.predict(tf_validation_set)
       print("predictions done, postprocessing predictions...")
       processed_pred = postprocess_function(dataset, data_tokenized,
pred["start_logits"], pred["end_logits"])
       print("postprocess done, evaluating metrics...")
       formatted_predictions = [
       {"id": k, "prediction_text": v, "no_answer_probability": 0.0} for k, v in
processed_pred.items()
       ]
       references = [{"id": ex["id"], "answers": ex["answers"]} for ex in
dataset]

       scores = metric.compute(predictions=formatted_predictions,
references=references)
       model_metrics[model_path] = scores
df = pd.DataFrame(model_metrics)
df = df.drop(["best_exact_thresh", "best_f1_thresh"])
df.to_csv("scores2.csv")
```