How To Train Model for Open Book Q&A Technique

In this notebook we demonstrate how to train a model to be used with top scoring Open Book Q&A method. The Open Book method was first presented by JJ (@jjinho) here, then Quangteo (@quangbk) improved RAM usage here, and Anil (@nlztrk) combined with Q&A here. Radek (@radek1) demonstrated the strength of Q&A here. Next Mgoksu (@mgoksu) demonstrated how to achieve top public LB=0.807 using this method here by finetuning DeBerta large on this method.

In order to train a model for use with Open Book Q&A, we need a CSV that contains; `prompt` **(i.e. question),** `A, B, C, D, E` **(i.e. answer choices), and we need a column of** `context` **extracted from wikipedia pages for each question. To generate the** `context` **column, we run Mgoksu's notebook here. In code cell #5, we load our CSV without** `context` **column with code** `trn = pd.read_csv(OUR_DATASET.CSV)`**. Then in code cell #21 our dataset is saved to disk as** `test_context.csv` **with the column** `context` **added.**

I have searched and concatenated all publicly shared datasets into one 60k CSV and then ran Mgoksu's notebook with `NUM_TITLES_INCLUDE = 5` **and** `NUM_SENTENCES_INCLUDE = 20`**. This added an additional** `context` **column. I uploaded the resultant CSV file to a Kaggle dataset here. If you enjoy the notebook you are reading, please upvote the dataset too. Thanks!**

(image source here)

Load CSV

We will load 60k CSV of `prompts`**,** `A,B,C,D,E`**, and** `context` **from my Kaggle dataset here. This dataset is all publicly shared datasets concatenated then processed with Mgoksu's notebook here to create a** `context` **column. (To learn more about the datasets within read my discussion post). This Kaggle dataset also contains competition** `train.csv` **with added** `context` **column (to be used as a validation dataset).**

In this train notebook, we have internet turned on and can choose whatever model we wish to download and train. After we finetune this model, we will create a second notebook with the Open Book Q&A technique and load the finetuned model from the output of this notebook. The second notebook will have internet turned off so that it can be submitted to Kaggle's competition.

In [1]:

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]="0,1"

from typing import Optional, Union
import pandas as pd, numpy as np, torch
from datasets import Dataset
from dataclasses import dataclass
from transformers import AutoTokenizer
from transformers import EarlyStoppingCallback
from transformers.tokenization_utils_base import PreTrainedTokenizerBase, PaddingStrategy
from transformers import AutoModelForMultipleChoice, TrainingArguments, Trainer

VER=2
# TRAIN WITH SUBSET OF 60K
NUM_TRAIN_SAMPLES = 1_024
# PARAMETER EFFICIENT FINE TUNING
# PEFT REQUIRES 1XP100 GPU NOT 2XT4
USE_PEFT = False
# NUMBER OF LAYERS TO FREEZE
# DEBERTA LARGE HAS TOTAL OF 24 LAYERS
FREEZE_LAYERS = 18
# BOOLEAN TO FREEZE EMBEDDINGS
FREEZE_EMBEDDINGS = True
# LENGTH OF CONTEXT PLUS QUESTION ANSWER
```

```
MAX_INPUT = 256
# HUGGING FACE MODEL
MODEL = 'microsoft/deberta-v3-large'
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy versi
on >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:

```
df_valid = pd.read_csv('/kaggle/input/60k-data-with-context-v2/train_with_context2.csv')
print('Validation data size:', df_valid.shape )
df_valid.head()
```

```
Validation data size: (200, 8)
```

Out[2]:

	prompt	context	A	B	C	D	E	answer
0	Which of the following statements accurately d...	The presence of a clustered thick disk-like co...	MOND is a theory that reduces the observed mis...	MOND is a theory that increases the discrepanc...	MOND is a theory that explains the missing bar...	MOND is a theory that reduces the discrepancy ...	MOND is a theory that eliminates the observed ...	D
1	Which of the following is an accurate definiti...	Many of these systems evolve in a self-similar...	Dynamic scaling refers to the evolution of sel...	Dynamic scaling refers to the non-evolution of...	Dynamic scaling refers to the evolution of sel...	Dynamic scaling refers to the non-evolution of...	Dynamic scaling refers to the evolution of sel...	A
2	Which of the following statements accurately d...	It is possible that this usage is related with...	The triskeles symbol was reconstructed as a fe...	The triskeles symbol is a representation of th...	The triskeles symbol is a representation of a ...	The triskeles symbol represents three interloc...	The triskeles symbol is a representation of th...	A
3	What is the significance of regularization in ...	Renormalization is distinct from regularizatio...	Regularizing the mass-energy of an electron wi...	Regularizing the mass-energy of an electron wi...	Regularizing the mass-energy of an electron wi...	Regularizing the mass-energy of an electron wi...	Regularizing the mass-energy of an electron wi...	C
4	Which of the following statements accurately d...	Several qualitative observations can be made o...	The angular spacing of features in the diffrac...	The angular spacing of features in the diffrac...	The angular spacing of features in the diffrac...	The angular spacing of features in the diffrac...	The angular spacing of features in the diffrac...	D

In [3]:

```
df_train = pd.read_csv('/kaggle/input/60k-data-with-context-v2/all_12_with_context2.csv')
df_train = df_train.drop(columns="source")
df_train = df_train.fillna('').sample(NUM_TRAIN_SAMPLES)
print('Train data size:', df_train.shape )
df_train.head()
```

```
Train data size: (1024, 8)
```

Out[3]:

	prompt	context	A	B	C	D	E	answer
34359	What is the likely function of salivarius-1 RNAs?	The salivarius-1 motif occurs in various strai...	They occur in various strains of Lactobacillus...	They function as cis-regulatory elements.	They function as small RNAs in trans.	They come from unknown species.	They are conserved RNA structures.	C
35048	What are the typical dimensions and characteri...	These include the ice streams with the greates...	Ice streams move at a relatively slow pace com...	Ice streams move slowly, at about a few inches...	Ice streams move rapidly, as fast as a hundred...	Ice streams move upwards of a mile per year, a...	Ice streams can move several feet per year and...	D
13866	What was the reason for the government	Soon after the attacks of September	The government detained	The government detained	The government detained	The government detained	The government detained	C

	prompt	context	A	B	C	D	E	answer
23322	What was Jimi Goodwin's role within the band D...	Jimi Goodwin (born Jamie Francis Alexander Goo...	Drummer.	Keyboardist.	Bassist and vocalist.	Lead vocalist.	Lead guitarist.	C
30714	What is the physical characteristic of the for...	The hindwings are whitish and tinged with brow...	The forewing has pale brown hair-pencils in th...	The forewing is greyish with irregular fasciae.	The forewing displays white spots near the lea...	The forewing has weak antemedial and post-medi...	The forewing has a slightly concave costa.	B

Data Loader

Code is from Radek's notebook here with modifications to the tokenization process.

In [4]:

```
option_to_index = {option: idx for idx, option in enumerate('ABCDE')}
index_to_option = {v: k for k,v in option_to_index.items()}

def preprocess(example):
    first_sentence = [ "[CLS] " + example['context'] ] * 5
    second_sentences = [" #### " + example['prompt'] + " [SEP] " + example[option] + " [
SEP]" for option in 'ABCDE']
    tokenized_example = tokenizer(first_sentence, second_sentences, truncation='only_fir
st',
                                  max_length=MAX_INPUT, add_special_tokens=False)
    tokenized_example['label'] = option_to_index[example['answer']]

    return tokenized_example

@dataclass
class DataCollatorForMultipleChoice:
    tokenizer: PreTrainedTokenizerBase
    padding: Union[bool, str, PaddingStrategy] = True
    max_length: Optional[int] = None
    pad_to_multiple_of: Optional[int] = None

    def __call__(self, features):
        label_name = 'label' if 'label' in features[0].keys() else 'labels'
        labels = [feature.pop(label_name) for feature in features]
        batch_size = len(features)
        num_choices = len(features[0]['input_ids'])
        flattened_features = [
            [{k: v[i] for k, v in feature.items()} for i in range(num_choices)] for feat
ure in features
        ]
        flattened_features = sum(flattened_features, [])

        batch = self.tokenizer.pad(
            flattened_features,
            padding=self.padding,
            max_length=self.max_length,
            pad_to_multiple_of=self.pad_to_multiple_of,
            return_tensors='pt',
        )
        batch = {k: v.view(batch_size, num_choices, -1) for k, v in batch.items()}
        batch['labels'] = torch.tensor(labels, dtype=torch.int64)
        return batch
```

In [5]:

```
tokenizer = AutoTokenizer.from_pretrained(MODEL)
dataset_valid = Dataset.from_pandas(df_valid)
dataset = Dataset.from_pandas(df_train)
dataset = dataset.remove_columns(["__index_level_0__"])
dataset
```

```
Special tokens have been added in the vocabulary, make sure the associated word embedding
s are fine-tuned or trained.
/opt/conda/lib/python3.10/site-packages/transformers/convert_slow_tokenizer.py:470: UserW
arning: The sentencepiece tokenizer that you are converting to a fast tokenizer uses the
byte fallback option which is not implemented in the fast tokenizers. In practice this me
ans that the fast version of the tokenizer can produce unknown tokens whereas the sentenc
epiece version would have converted these unknown tokens into a sequence of byte tokens m
atching the original piece of text.
  warnings.warn(
Special tokens have been added in the vocabulary, make sure the associated word embedding
s are fine-tuned or trained.
```

Out[5]:

```
Dataset({
    features: ['prompt', 'context', 'A', 'B', 'C', 'D', 'E', 'answer'],
    num_rows: 1024
})
```

In [6]:

```
tokenized_dataset_valid = dataset_valid.map(preprocess, remove_columns=['prompt', 'contex
t', 'A', 'B', 'C', 'D', 'E', 'answer'])
tokenized_dataset = dataset.map(preprocess, remove_columns=['prompt', 'context', 'A', 'B'
, 'C', 'D', 'E', 'answer'])
tokenized_dataset
```

Out[6]:

```
Dataset({
    features: ['input_ids', 'token_type_ids', 'attention_mask', 'label'],
    num_rows: 1024
})
```

Build Model

We will use a Hugging Face AutoModelForMultipleChoice. For the list of possible models, see Hugging Face's repository [here](). We can optionally use PEFT to accelerate training and use less memory. However i have noticed that validation accuracy is less. (Note that PEFT requires us to use 1xP100 not 2xT4 GPU. I'm not sure why). We can also optionally freeze layers. This also accelerates training and uses less memory. However validation accuracy may become less.

In [7]:

```
model = AutoModelForMultipleChoice.from_pretrained(MODEL)
```

```
Some weights of DebertaV2ForMultipleChoice were not initialized from the model checkpoint
at microsoft/deberta-v3-large and are newly initialized: ['pooler.dense.bias', 'classifie
r.weight', 'classifier.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predi
ctions and inference.
```

In [8]:

```
# NOTE PEFT REQUIRES US TO USE 1XP100 NOT 2XT4. I'M NOT SURE WHY.
if USE_PEFT:
    !pip install --no-index --no-deps /kaggle/input/llm-whls/peft-0.4.0-py3-none-any.whl
```

In [9]:

```
if USE_PEFT:
    print('We are using PEFT.')
    from peft import LoraConfig, get_peft_model, TaskType
    peft_config = LoraConfig(
        r=8, lora_alpha=4, task_type=TaskType.SEQ_CLS, lora_dropout=0.1,
```

```
        bias="none", inference_mode=False,
        target_modules=["query_proj", "value_proj"],
        modules_to_save=['classifier','pooler'],
    )
    model = get_peft_model(model, peft_config)
    model.print_trainable_parameters()
```

In [10]:

```
if FREEZE_EMBEDDINGS:
    print('Freezing embeddings.')
    for param in model.deberta.embeddings.parameters():
        param.requires_grad = False
if FREEZE_LAYERS>0:
    print(f'Freezing {FREEZE_LAYERS} layers.')
    for layer in model.deberta.encoder.layer[:FREEZE_LAYERS]:
        for param in layer.parameters():
            param.requires_grad = False
```

```
Freezing embeddings.
Freezing 18 layers.
```

MAP@3 Metric

The competition metric is MAP@3 therefore we will make a custom code to add to Hugging Face's trainer. Discussion [here]

In [11]:

```
def map_at_3(predictions, labels):
    map_sum = 0
    pred = np.argsort(-1*np.array(predictions),axis=1)[:,:3]
    for x,y in zip(pred,labels):
        z = [1/i if y==j else 0 for i,j in zip([1,2,3],x)]
        map_sum += np.sum(z)
    return map_sum / len(predictions)

def compute_metrics(p):
    predictions = p.predictions.tolist()
    labels = p.label_ids.tolist()
    return {"map@3": map_at_3(predictions, labels)}
```

Train and Save

We will now train and save our model using Hugging Face's easy to use trainer. By adjusting the parameters in this notebook, we can achieve `CV MAP@3 = 0.915+` and corresponding single model `LB MAP@3 = 0.830+` wow!

In we run this notebook outside of Kaggle then we can train longer and with more RAM. If we run this notebook on Kaggle, then we need to use tricks to train models efficiently. Here are some ideas:

- **use fp16 (this speeds up T4 not P100)**
- **use gradient_accumlation_steps (this simulates larger batch sizes)**
- **use gradient_checkpointing (this uses disk to save RAM)**
- **use 2xT4 instead of 1xP100 (this doubles GPUs)**
- **freeze model embeddings (this reduces weights to train)**
- **freeze some model layers (this reduces weights to train)**
- **use PEFT (this reduces weights to train)**
- **increase LR and decrease epochs (this reduces work)**
- **use smaller models (this reduces weights to train)**

In [12]:

```
training_args = TrainingArguments(
```

```
    warmup_ratio=0.1,
    learning_rate=2e-5,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=2,
    num_train_epochs=2,
    report_to='none',
    output_dir = f'./checkpoints_{VER}',
    overwrite_output_dir=True,
    fp16=True,
    gradient_accumulation_steps=8,
    logging_steps=25,
    evaluation_strategy='steps',
    eval_steps=25,
    save_strategy="steps",
    save_steps=25,
    load_best_model_at_end=False,
    metric_for_best_model='map@3',
    lr_scheduler_type='cosine',
    weight_decay=0.01,
    save_total_limit=2,
)
```

In [13]:

```
trainer = Trainer(
    model=model,
    args=training_args,
    tokenizer=tokenizer,
    data_collator=DataCollatorForMultipleChoice(tokenizer=tokenizer),
    train_dataset=tokenized_dataset,
    eval_dataset=tokenized_dataset_valid,
    compute_metrics = compute_metrics,
    #callbacks=[EarlyStoppingCallback(early_stopping_patience=5)],
)

trainer.train()
trainer.save_model(f'model_v{VER}')
```

```
You're using a DebertaV2TokenizerFast tokenizer. Please note that with a fast tokenizer,
using the `__call__` method is faster than using a method to encode the text followed by
a call to the `pad` method to get a padded encoding.
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning:
Was asked to gather along dimension 0, but all input tensors were scalars; will instead u
nsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
```

[128/128 23:01, Epoch 2/2]

Step	Training Loss	Validation Loss	Map@3
25	1.612600	1.608748	0.610833
50	1.590500	1.501639	0.763333
75	1.211200	1.223504	0.749167
100	0.987000	1.058218	0.811667
125	0.944000	1.054115	0.810000

```
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning:
Was asked to gather along dimension 0, but all input tensors were scalars; will instead u
nsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning:
Was asked to gather along dimension 0, but all input tensors were scalars; will instead u
nsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning:
Was asked to gather along dimension 0, but all input tensors were scalars; will instead u
nsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning:
```

```
Was asked to gather along dimension 0, but all input tensors were scalars; will instead u
nsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning:
Was asked to gather along dimension 0, but all input tensors were scalars; will instead u
nsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
```

Verify Saved Model

During training, we see the MAP@3 validation score above. Let's load the saved model and compute it again here to verify that our model is saved correctly.

In [14]:

```
del model, trainer
if USE_PEFT:
    model = AutoModelForMultipleChoice.from_pretrained(MODEL)
    model = get_peft_model(model, peft_config)
    checkpoint = torch.load(f'model_v{VER}/pytorch_model.bin')
    model.load_state_dict(checkpoint)
else:
    model = AutoModelForMultipleChoice.from_pretrained(f'model_v{VER}')
trainer = Trainer(model=model)
```

In [15]:

```
test_df = pd.read_csv('/kaggle/input/60k-data-with-context-v2/train_with_context2.csv')
tokenized_test_dataset = Dataset.from_pandas(test_df).map(
        preprocess, remove_columns=['prompt', 'context', 'A', 'B', 'C', 'D', 'E'])

test_predictions = trainer.predict(tokenized_test_dataset).predictions
predictions_as_ids = np.argsort(-test_predictions, 1)
predictions_as_answer_letters = np.array(list('ABCDE'))[predictions_as_ids]
predictions_as_string = test_df['prediction'] = [
    ' '.join(row) for row in predictions_as_answer_letters[:, :3]
]
```

```
/opt/conda/lib/python3.10/site-packages/torch/nn/parallel/_functions.py:68: UserWarning:
Was asked to gather along dimension 0, but all input tensors were scalars; will instead u
nsqueeze and return a vector.
  warnings.warn('Was asked to gather along dimension 0, but all '
```

Compute Validation Score

In [16]:

```
# https://www.kaggle.com/code/philippsinger/h2ogpt-perplexity-ranking
import numpy as np
def precision_at_k(r, k):
    """Precision at k"""
    assert k <= len(r)
    assert k != 0
    return sum(int(x) for x in r[:k]) / k

def MAP_at_3(predictions, true_items):
    """Score is mean average precision at 3"""
    U = len(predictions)
    map_at_3 = 0.0
    for u in range(U):
        user_preds = predictions[u].split()
        user_true = true_items[u]
        user_results = [1 if item == user_true else 0 for item in user_preds]
        for k in range(min(len(user_preds), 3)):
            map_at_3 += precision_at_k(user_results, k+1) * user_results[k]
    return map_at_3 / U
```

In [17]:

```
m = MAP_at_3(test_df.prediction.values, test_df.answer.values)
print( 'CV MAP@3 =',m )
```

```
CV MAP@3 = 0.81
```