fine_tune_bert_on_custom_dataset

December 28, 2023

1 FINE TUNE BERT WITH CUSTOM DATASET

1.1 CONNECT DRIVER

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]:
```
%cd /content/drive/MyDrive/Colab_Notebooks/nlp
```

```
/content/drive/MyDrive/Colab_Notebooks/nlp
```

1.2 INTRODUCE OPTIMIZE HUGGING FACE MODEL WEIGHTS AND BIASES

2 Optimize Hugging Face models with Weights & Biases

Hugging Face provides tools to quickly train neural networks for NLP (Natural Language Processing) on any task (classification, translation, question answering, etc) and any dataset with PyTorch and TensorFlow 2.0.

Coupled with Weights & Biases integration, you can quickly train and monitor models for full traceability and reproducibility without any extra line of code! You just need to install the library, sign in, and your experiments will automatically be logged:

```
pip install wandb
wandb login
```

Note: To enable logging to W&B, set `report_to` to `wandb` in your `TrainingArguments` or script.

W&B integration with Hugging Face can automatically: * log your configuration parameters * log your losses and metrics * log gradients and parameter distributions * log your model * keep track of your code * log your system metrics (GPU, CPU, memory, temperature, etc)

2.1 INSTALLATION

[ ]:
```
!pip install datasets wandb evaluate accelerate -qU
!wget https://raw.githubusercontent.com/huggingface/transformers/master/
↪examples/pytorch/text-classification/run_glue.py
```

[ ]:
```
# the run_glue.py script requires transformers dev
!pip install -q git+https://github.com/huggingface/transformers
```

We finally make sure we're logged into W&B so that our experiments can be associated to our account.

[5]:
```
import wandb
```

[6]:
```
wandb.login()
```

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

```
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server
locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here:
https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to
quit:

 ..........

wandb: Appending key for api.wandb.ai to your netrc file:
/root/.netrc
```

[6]:
```
True
```

2.2 Configuration tips

W&B integration with Hugging Face can be configured to add extra functionalities:

- auto-logging of models as artifacts: just set environment varilable `WANDB_LOG_MODEL` to `true`
- log histograms of gradients and parameters: by default gradients are logged, you can also log parameters by setting environment variable `WANDB_WATCH` to `all`
- set custom run names with `run_name` arg present in scripts or as part of `TrainingArguments`
- organize runs by project with the `WANDB_PROJECT` environment variable

For more details refer to W&B + HF integration documentation.

Let's log every trained model.

[7]:
```
%env WANDB_LOG_MODEL=true
```

```
env: WANDB_LOG_MODEL=true
```

2.3 Training a new model the quick way!

When working on a new problem, you should always check the summary of task as there will often be a script that can already solve your task. At a minimum they will be a great source of inspiration for your own custom pipeline.

Let's use the Hugging Face script responsible for training on any GLUE task, such as sequence classification.

[8]:
```
!wget https://raw.githubusercontent.com/huggingface/transformers/master/
↪examples/pytorch/text-classification/run_glue.py
```

```
--2023-12-28 03:06:24--  https://raw.githubusercontent.com/huggingface/transform
ers/master/examples/pytorch/text-classification/run_glue.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)…
185.199.108.133, 185.199.109.133, 185.199.111.133, …
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 28335 (28K) [text/plain]
Saving to: 'run_glue.py.1'

run_glue.py.1       100%[===================>]  27.67K  --.-KB/s    in 0.002s

2023-12-28 03:06:25 (11.3 MB/s) - 'run_glue.py.1' saved [28335/28335]
```

These scripts are automatically instrumented with logging when `wandb` is installed and logged in.

Just set `report_to` to `wandb` to enable logging through W&B.

Note: This cell can take up to 5 minutes to run.

[ ]:
```
!python run_glue.py \
  --report_to wandb \
  --model_name_or_path bert-base-uncased \
  --task_name MRPC \
  --learning_rate 1e-4 \
  --do_train \
  --do_eval \
  --max_steps 300 \
  --logging_steps 30 \
  --evaluation_strategy steps \
  --output_dir /tmp/MRPC \
  --overwrite_output_dir \
  --run_name demo
```

3 Advanced usage & custom training

Let's create our own logic for a more customized training.

3.0.1 Preparing a dataset

The dataset will vary based on the task you work on. Let's work on sequence classification!

Our dataset will be composed of sentences and their associated classes. For example if you wanted to identify the subject of a conversation, you could create a dataset such as:

input	class
hôm nay tôi buồn	negative
món ăn này ngon quá	positive
tớ rất là chán cậu luôn ấy !	negative

The objective of our trained model will be to correctly identify the class associated to new sentences.

[52]:
```
from datasets import Dataset
import pandas as pd
from sklearn.model_selection import train_test_split
```

[53]:
```
data_1 = pd.read_csv("preprocess_train.csv")
data_2 = pd.read_csv("NTC_SV_test.csv")
data_3 = pd.read_csv("NTC_SV_train.csv")
pd_data = pd.concat([data_1, data_2, data_3], axis=0)
```

[54]:
```
pd_data = pd_data.dropna()
```

[55]:
```
pd_data = pd_data[['review', 'label']]
```

[56]:
```
pd_data
```

[56]:
```
                                                review  label
0      dung được positive sản_phẩm tốt positive cam o…      1
1      chất_lượng sản_phẩm tuyệt_vời positive mịn pos…      1
2      chất_lượng sản_phẩm tuyệt_vời positive nhưng k…      1
3      negative mình hơi thất_vọng negative chút vì m…      0
4      lần trước mình mua áo_gió màu hồng positive rấ…      0
…                                                  …    …
40756                                            thiếu      0
40757                                              xấu      0
40758                                               ẩu      0
40759                                              lộn      0
40760                                         hoang_sơ      0

[82928 rows x 2 columns]
```

[57]:
```
pd_train, pd_test = train_test_split(pd_data,
                                     test_size=0.2,
                                     random_state=42)
```

[58]:
```
pd_train.reset_index(drop=True, inplace=True)
```

[59]:
```
pd_test.reset_index(drop=True, inplace=True)
```

```
[60]: data_train = Dataset.from_pandas(pd_train)
      data_test = Dataset.from_pandas(pd_test)
```

```
[61]: data_train
```

```
[61]: Dataset({
          features: ['review', 'label'],
          num_rows: 66342
      })
```

```
[62]: data_test
```

```
[62]: Dataset({
          features: ['review', 'label'],
          num_rows: 16586
      })
```

We can easily access any element.

```
[65]: data_train[0]
```

```
[65]: {'review': 'nhin anh da ro bo sach can duoc positive bao boc can than positive
      hon truoc khi dem chuyen phat than',
       'label': 0}
```

`str2int` and `int2str` help us go from class label to their integer mapping.

For our topic classification task, we use `question_title` as input and try to predict `topic`.

```
[69]: label_list = data_train.unique('label')
      label_list.sort()
      label_list
```

```
[69]: [0, 1]
```

This particular dataset is split between 10 different topics, that will be represented by 10 classes from our model output.

```
[70]: num_labels = len(label_list)
      num_labels
```

```
[70]: 2
```

The “topic” class needs to be renamed to “labels” for the `Trainer` to find it.

```
[72]: data_train = data_train.rename_column('label', 'labels')
      data_test = data_test.rename_column('label', 'labels')
```

3.0.2 Tokenizing the dataset

In order to train a neural network, we need to convert our inputs to numbers: * the tokenizer divides a sequence of characters into tokens, ie sub-sequences (such as words, characters, sub-words…) * each unique token is mapped to a unique integer

There are many types of tokenizers. Transformers can auto-select the right `Tokenizer` associated to a specific model.

[73]:
```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained('distilbert-base-uncased')
```

```
tokenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]

config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
```

The tokenizer let us quickly preprocess our data.

[75]:
```
sample_input = data_train[0]['review']
sample_input
```

[75]:
```
'nhin anh da ro bo sach can duoc positive bao boc can than positive hon truoc
khi dem chuyen phat than'
```

[76]:
```
tokenizer(sample_input)
```

[76]:
```
{'input_ids': [101, 18699, 2378, 2019, 2232, 4830, 20996, 8945, 17266, 2232,
2064, 6829, 2278, 3893, 25945, 8945, 2278, 2064, 2084, 3893, 10189, 19817,
19098, 2278, 1047, 4048, 17183, 14684, 20684, 6887, 4017, 2084, 102],
'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

The tokenizer can quickly process an entire dataset and cache the results locally to avoid any future tokenization of the same data.

We leverage `dataset.map(fn)` function which can efficiently apply any function to a dataset. We also take advantage of batch processing which is supported by the tokenizer and makes the operation even faster.

[77]:
```
data_train = data_train.map(lambda x: tokenizer(x['review'],
                                                truncation=True),
                            batched=True)
data_test = data_test.map(lambda x: tokenizer(x['review'],
                                              truncation=True),
                          batched=True)
```

```
Map:   0%|          | 0/66342 [00:00<?, ? examples/s]

Map:   0%|          | 0/16586 [00:00<?, ? examples/s]
```

We truncate the data to the max length supported by the model. During training, we will pass the tokenizer to pad inputs to the longest sequence of the batch (the model requires same length inputs in a single batch).

Our dataset now contains new keys: `input_ids` (tokens) and `attention_mask` (needed for certain models).

[ ]:
```
data_train[0]
```

3.0.3 Loading a model

Plenty of models are available and can be explored on the Model Hub.

Once a model has been selected, it can be automatically loaded and adapted to one of its supported tasks.

[79]:
```
from transformers import AutoModelForSequenceClassification
model = AutoModelForSequenceClassification.
 ↪from_pretrained('distilbert-base-uncased',                           ␣
 ↪num_labels=num_labels)
```

```
model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]

Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['pre_classifier.weight', 'classifier.bias', 'classifier.weight',
'pre_classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

In this case, we are loading a pre-trained network to which a custom head has been added for sequence classification and presents 10 classes corresponding to the possible topics of this dataset.

Let's make a function to return the topic prediction from a sample question.

[96]:
```
import torch

def get_topic(sentence, tokenize=tokenizer, model=model):
    # tokenize the input
    inputs = tokenizer(sentence, return_tensors='pt')
    # ensure model and inputs are on the same device (GPU)
    inputs = {name: tensor.cuda() for name, tensor in inputs.items()}
    model = model.cuda()
    # get prediction - 10 classes "probabilities"
    # (not really true because they still need to be normalized)
    with torch.no_grad():
        predictions = model(**inputs)[0].cpu().numpy()
    # get the top prediction class and convert it to its associated label
    top_prediction = predictions.argmax().item()
    return top_prediction
```

```
    # return data_train.features['labels'].int2str(top_prediction)
```

Let's test a prediction on a sample sentence.

[97]:
```
get_topic('positive')
```

[97]: 0

Obviously the model has not been trained yet so the results are still random.

3.0.4 Training the model

We now need to fine-tune the model based on our dataset.

The `Trainer` class let us easily train a model and is very flexible.

Note: set `report_to` to `wandb` in `TrainingArguments` to enable logging through W&B.

[98]:
```
from transformers import Trainer, TrainingArguments

args = TrainingArguments(
    # enable logging to W&B
    report_to = 'wandb',
    # output directory
    output_dir = 'topic_classification',
    overwrite_output_dir = True,
    # check evaluation metrics at each epoch
    evaluation_strategy = 'steps',
    # we can customize learning rate
    learning_rate = 5e-5,
    max_steps = 30000,
    # we will log every 100 steps
    logging_steps = 100,
    # we will perform evaluation every 500 steps
    eval_steps = 5000,
    save_steps = 10000,
    load_best_model_at_end = True,
    metric_for_best_model = 'accuracy',
    # name of the W&B run
    run_name = 'custom_training'
)
```

For more customization, refer to `TrainingArguments` documentation.

We can optionally define metrics to calculate in addition to the loss through the `compute_metrics` function.

Several metrics are readily available from the datasets library to monitor model performance.

[99]:
```python
from datasets import load_metric
import numpy as np

accuracy_metric = load_metric("accuracy")

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions,
                            axis=1)
    # metrics from the datasets library
    # have a `compute` method
    return accuracy_metric.compute(predictions=predictions,
                                   references=labels)
```

```
<ipython-input-99-d14036c6f9ef>:4: FutureWarning: load_metric is deprecated and
will be removed in the next major version of datasets. Use 'evaluate.load'
instead, from the new library  Evaluate: https://huggingface.co/docs/evaluate
  accuracy_metric = load_metric("accuracy")
/usr/local/lib/python3.10/dist-packages/datasets/load.py:752: FutureWarning: The
repository for accuracy contains custom code which must be executed to correctly
load the metric. You can inspect the repository content at https://raw.githubuse
rcontent.com/huggingface/datasets/2.16.0/metrics/accuracy/accuracy.py
You can avoid this message in future by passing the argument
`trust_remote_code=True`.
Passing `trust_remote_code=True` will be mandatory to load this metric from the
next major release of `datasets`.
  warnings.warn(
```

```
Downloading builder script:   0%|          | 0.00/1.65k [00:00<?, ?B/s]
```

The `Trainer` handles all the training & evaluation logic.

[100]:
```python
trainer = Trainer(
    model = model,                    # model to be trained
    args = args,                      # training args
    train_dataset=data_train,
    eval_dataset=data_test,
    tokenizer=tokenizer,              # for padding batched data
    compute_metrics=compute_metrics # for custom metrics
)
```

We can verify that we initially have an accuracy of about 10% (random predictions over 10 classes).

[101]:
```python
trainer.evaluate()
```

```
You're using a DistilBertTokenizerFast 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.
```

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

<IPython.core.display.HTML object>

wandb: Currently logged in as: nvv2023. Use `wandb
login --relogin` to force relogin

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

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

[101]:
```
{'eval_loss': 0.6954892873764038,
 'eval_accuracy': 0.4697938020016882,
 'eval_runtime': 166.5291,
 'eval_samples_per_second': 99.598,
 'eval_steps_per_second': 12.454}
```

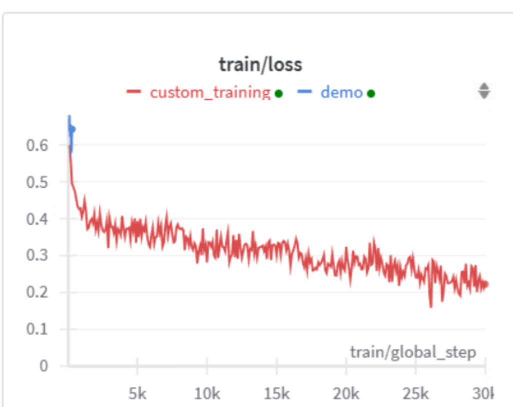
We start training by simply calling `train()`.

[102]:
```python
trainer.train()
```

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

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

[102]:
```
TrainOutput(global_step=30000, training_loss=0.31151498403549194,
metrics={'train_runtime': 8120.5207, 'train_samples_per_second': 29.555,
'train_steps_per_second': 3.694, 'total_flos': 1.911619333188197e+16,
'train_loss': 0.31151498403549194, 'epoch': 3.62})
```



We can now use the trained model for better predictions.

[104]:
```python
get_topic('cái này không ngon mày ơi!')
```

[104]: 0

When we want to close our W&B run, we can call `wandb.finish()` (mainly useful in notebooks, called automatically in scripts).

[106]:
```python
wandb.finish()
```

```
VBox(children=(Label(value='256.342 MB of 256.351 MB uploaded\r'),␣
↪FloatProgress(value=0.9999664326761674, max…
```

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

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

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

4 load model from checkpoint

5 Application

[111]:
```python
from transformers import AutoTokenizer, AutoModelForSequenceClassification

tokenizer_load = AutoTokenizer.from_pretrained("/content/drive/MyDrive/
↪Colab_Notebooks/nlp/topic_classification/checkpoint-30000")
model_load = AutoModelForSequenceClassification.from_pretrained("/content/drive/
↪MyDrive/Colab_Notebooks/nlp/topic_classification/checkpoint-30000")
```

[115]:
```python
text_to_predict = "Cái này không ngon lắm bạn ơi"

# Tokenize văn bản
inputs_load = tokenizer_load(text_to_predict,
                             return_tensors="pt")

# Dự đoán
with torch.no_grad():
    outputs = model_load(**inputs_load)

# Lấy dự đoán
predictions = outputs.logits
```

[116]:
```python
predictions
```

[116]: tensor([[1.9997, -1.3276]])

[117]:
```python
import torch
from torch.nn import functional as F
```

```python
# Áp dụng softmax để có xác suất
probs = F.softmax(predictions, dim=1)

# Lấy nhãn có xác suất cao nhất
predicted_class = torch.argmax(probs, dim=1).item()

print(f"Nhãn dự đoán: {predicted_class}")
```

```
Nhãn dự đoán: 0
```