Final Project Report-Team 6

October 23, 2023

[ ]:

```python
PROJECT = "aai520-project" # @param {type:"string"}
GROUP = "aai520-group6" # @param {type:"string"}
MODELS_STR = "distilbert-base-uncased, bert-base-uncased, roberta-base" # @param {type:"string"}
MODELS = [model.strip() for model in MODELS_STR.split(',')]
TASK = "question-answering" # @param {type:"string"}
DATASET = "squad_v2" # @param {type:"string"}
NUM_TRAIN_EPOCHS = 4  # @param {type:"integer"}
LEARNING_RATE = 2e-5  # @param
GRADIENT_ACCUMULATION_STEPS = 4  # @param {type:"integer"}
PER_DEVICE_TRAIN_BATCH_SIZE = 128  # @param {type:"integer"}
PER_DEVICE_EVAL_BATCH_SIZE = 128  # @param {type:"integer"}
EVALUATION_STRATEGY = 'steps'  # @param {type:"string"}
EVAL_STEPS = 100  # @param {type:"integer"}
SAVE_STRATEGY = "steps"  # @param {type:"string"}
SAVE_STEPS = 100  # @param {type:"integer"}
LOGGING_STEPS = 100  # @param {type:"integer"}
FP16 = True  # @param {type:"boolean"}
DATALOADER_NUM_WORKERS = 2  # @param {type:"integer"}
REPORT_TO = 'tensorboard'  # @param {type:"string"}
LOAD_BEST_MODEL_AT_END = True  # @param {type:"boolean"}
DISABLE_TQDM = False  # @param {type:"boolean"}
PUSH_TO_HUB = True  # @param {type:"boolean"}
OVERWRITE_OUTPUT_DIRECTORY = True  # @param {type:"boolean"}
```

[ ]:

```python
#------------------------------------------------------------------------------#
# CLEAN ENVIRONMENT
#------------------------------------------------------------------------------#

import gc
gc.collect()

import torch
torch.cuda.empty_cache()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#------------------------------------------------------------------------------#
```

```python
# MOUNT GOOGLE DRIVE
#------------------------------------------------------------------------------#

from pathlib import Path
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
DRIVE_PATH = Path("/content/drive/My Drive/Colab Notebooks")

#------------------------------------------------------------------------------#
# HANDLE PATHS
#------------------------------------------------------------------------------#

PROJECT_PATH = DRIVE_PATH / PROJECT
PROJECT_PATH.mkdir(parents=True, exist_ok=True)
CHECKPOINTS_DIR = PROJECT_PATH / "checkpoints"
CHECKPOINTS_DIR.mkdir(parents=True, exist_ok=True)
LOGS_DIR = PROJECT_PATH / "logs"
LOGS_DIR.mkdir(parents=True, exist_ok=True)
VISUALS_DIR = PROJECT_PATH / "visuals"
VISUALS_DIR.mkdir(parents=True, exist_ok=True)

#------------------------------------------------------------------------------#
# SYMLINK FOR FASTER FILESYSTEM TRAVERSAL
#------------------------------------------------------------------------------#

SHORTCUT = Path("/content/project")
if not SHORTCUT.exists():
    SHORTCUT.symlink_to(PROJECT_PATH)

#------------------------------------------------------------------------------#
# INSTALL DEPENDENCIES
#------------------------------------------------------------------------------#

!pip install \
    -qqq \
    --progress-bar=off \
    datasets \
    evaluate \
    huggingface_hub \
    python-dotenv \
    tensorboardcolab \
    transformers[torch] \
    txtai

#------------------------------------------------------------------------------#
# IMPORT LIBRARIES & SETUP/CONFIGURE
#------------------------------------------------------------------------------#
```

```
import json
import os

import evaluate
from datasets import load_dataset
from dotenv import load_dotenv
from evaluate.visualization import radar_plot
from huggingface_hub import EvalResult, ModelCard, ModelCardData, RepoCard,␣
  ↪login
from transformers.trainer_utils import get_last_checkpoint
from txtai.pipeline import HFTrainer



# LOGIN TO HUB
load_dotenv(dotenv_path=PROJECT_PATH / ".env")
login(token=os.getenv('HF_TOKEN'))

# IGNORE TRANSFORMER WARNINGS
os.environ['TRANSFORMERS_NO_ADVISORY_WARNINGS'] = 'true'

#------------------------------------------------------------------------------#
# LOAD DATASET
#------------------------------------------------------------------------------#

# DATASET
ds = load_dataset(DATASET)
ds.push_to_hub(repo_id=f"{GROUP}/{DATASET}")

#------------------------------------------------------------------------------#
# ITERATE THROUGH ALL MODELS
#------------------------------------------------------------------------------#

combined_results = {}

for pretrained_model in MODELS:

    # RENAME MODEL TO DISTINGUISH FINETUNED MODEL
    finetuned_model = pretrained_model.replace('base', 'finetuned')
    print(f'FINE-TUNING: {finetuned_model}')

    # CONSTRUCT STANDARD IDENTIFER FOR PUBLISHING
    IDENTIFIER = f"{GROUP}/{finetuned_model}-{DATASET}"

    # LOAD CHECKPOINTS
    model_checkpoints = CHECKPOINTS_DIR / finetuned_model
    model_checkpoints.mkdir(parents=True, exist_ok=True)
    last_checkpoint = get_last_checkpoint(model_checkpoints)

#------------------------------------------------------------------------------#
# TRAIN MODEL
#------------------------------------------------------------------------------#

    # use txtai for preconfigured preprocessing of SQuAD datasets
    # https://github.com/neuml/txtai
    trainer = HFTrainer()
    model, tokenizer = trainer(

        # SPECIFICATIONS
        base=last_checkpoint or pretrained_model,
        task=TASK,

        # DATASETS
        train=ds["train"],
        validation=ds["validation"],

        # HYPERPARAMETERS
        num_train_epochs=NUM_TRAIN_EPOCHS,
        learning_rate=LEARNING_RATE,
        gradient_accumulation_steps=GRADIENT_ACCUMULATION_STEPS,
        per_device_train_batch_size=PER_DEVICE_TRAIN_BATCH_SIZE,
        per_device_eval_batch_size=PER_DEVICE_EVAL_BATCH_SIZE,

        # CHECKPOINTS/OUTPUTS
        checkpoint=last_checkpoint,
        output_dir=str(model_checkpoints),

        # STRATEGIES
        evaluation_strategy=EVALUATION_STRATEGY,
        eval_steps=EVAL_STEPS,
        save_strategy=SAVE_STRATEGY,
        save_steps=SAVE_STEPS,
        logging_steps=LOGGING_STEPS,

        # OPTIMIZATIONS
        fp16=FP16,
        dataloader_num_workers=DATALOADER_NUM_WORKERS,

        # SAVING/LOGGING
        report_to=REPORT_TO,
        logging_dir=str(LOGS_DIR),
        load_best_model_at_end=LOAD_BEST_MODEL_AT_END,
        disable_tqdm=DISABLE_TQDM,
        # PUBLISH TO HUB
        push_to_hub=PUSH_TO_HUB,
        hub_model_id=IDENTIFIER,
        hub_token=os.getenv('HF_TOKEN')

    )

#------------------------------------------------------------------------------#
# VALIDATE MODEL
#------------------------------------------------------------------------------#

    print(f'VALIDATING: {finetuned_model}')
    evaluate.logging.set_verbosity_info()
    results = evaluate.evaluator("question-answering").compute(
        model_or_pipeline=IDENTIFIER,
        data=ds["validation"].select(range(2)),
        metric=DATASET,
        squad_v2_format=True,
    )

    for metric_type, metric_value  in results.items():
        metric_name = metric_type.upper() if metric_type == 'f1' else metric_type.capitalize()
        evaluate.push_to_hub(
            model_id=IDENTIFIER,
            metric_value=metric_value,
            metric_type=metric_type,
            metric_name=metric_name,
            dataset_type=DATASET,
            dataset_name="SQuAD v2",
            dataset_split="validation",
            task_type="question-answering",
            task_name="Question Answering",
            overwrite=True
        )

    # SAVE METRICS
    metrics_file_path = model_checkpoints / 'metrics.json'
    with open(metrics_file_path, 'w') as f:
        json.dump(results, f, indent=4)

    combined_results[pretrained_model] = results

#------------------------------------------------------------------------------#
# CREATE MODEL CARD FOR HUGGINGFACE
#------------------------------------------------------------------------------#
```

```
    # ITERATE AND ADD ALL METRICS
    eval_results = [
        EvalResult(
            task_type=TASK,
            dataset_type=DATASET,
            dataset_name='SQuAD v2',
            metric_type=metric_name.replace('_', ' ').title(),
            metric_value=metric_value
        )
        for metric_name, metric_value in results.items()
    ]

    # CREATE MODEL CARD DATA
    model_card_data = ModelCardData(
        language='en',
        license='mit',
        model_name=finetuned_model,
        eval_results=eval_results,

        model_details=f"""
        ## Abstract
        This model, '{finetuned_model}', is a question-answering chatbot trained on the SQuAD dataset, demonstrating competency in building conversational AI using recent advances in natural language processing. It utilizes a BERT model fine-tuned for extractive question answering.

        ## Data Collection and Preprocessing
        The model was trained on the Stanford Question Answering Dataset (SQuAD), which contains over 100,000 question-answer pairs based on Wikipedia articles. The data preprocessing involved tokenizing context paragraphs and questions, truncating sequences to fit BERT's max length, and adding special tokens to mark question and paragraph segments.

        ## Model Architecture and Training
        The architecture is based on the BERT transformer model, which was pretrained on large unlabeled text corpora. For this project, the BERT base model was fine-tuned on SQuAD for extractive question answering, with additional output layers for predicting the start and end indices of the answer span.

        ## SQuAD 2.0 Dataset
        SQuAD 2.0 combines the existing SQuAD data with over 50,000 unanswerable questions written adversarially by crowdworkers to look similar to answerable ones. This version of the dataset challenges models to not only produce answers when possible but also determine when no answer is supported by the paragraph and abstain from answering.
        """,
        intended_use=f"""
        - Answering questions from the {DATASET} dataset.
        - Developing question-answering systems within the scope of the␣
↪{PROJECT}.
        - Research and experimentation in the NLP question-answering domain.
        """,
        limitations_and_bias=f"""
        The model inherits limitations and biases from the '{pretrained_model}'␣
↪model, as it was trained on the same foundational data.
        It may underperform on questions that are ambiguous or too far outside␣
↪the scope of the topics covered in the {DATASET} dataset.
        Additionally, the model may reflect societal biases present in its␣
↪training data.
        """,
        ethical_considerations=f"""
        This model should not be used for making critical decisions without␣
↪human oversight,
        as it can generate incorrect or biased answers, especially for topics␣
↪not covered in the training data.
        Users should also consider the ethical implications of using AI in␣
↪decision-making processes and the potential for perpetuating biases.
        """,
        evaluation=f"""
        The model was evaluated on the {DATASET} dataset using various metrics.␣
↪These metrics, along with their corresponding scores,
        are detailed in the 'eval_results' section. The evaluation process␣
↪ensured a comprehensive assessment of the model's performance
        in question-answering scenarios.
        """,
        training=f"""
        The model was trained over {NUM_TRAIN_EPOCHS} epochs with a learning␣
↪rate of {LEARNING_RATE}, using a batch size of {PER_DEVICE_TRAIN_BATCH_SIZE}.
        The training utilized a cross-entropy loss function and the AdamW␣
↪optimizer, with gradient accumulation over {GRADIENT_ACCUMULATION_STEPS}␣
↪steps.
        """,
        tips_and_tricks=f"""
        For optimal performance, questions should be clear, concise, and␣
↪grammatically correct.
        The model performs best on questions related to topics covered in the␣
↪{DATASET} dataset.
        It is advisable to pre-process text for consistency in encoding and␣
↪punctuation, and to manage expectations for questions on topics outside the␣
↪training data.
        """
```

```
    )

    # CREATE AND SAVE MODEL CARD
    model_card = ModelCard.from_template(model_card_data)
    model_card_path = model_checkpoints / 'README.md'
    model_card.save(model_card_path)

#------------------------------------------------------------------------------#
# SAVE AND PUBLISH FINAL MODEL COMPONENTS
#------------------------------------------------------------------------------#

    # MODEL
    model.save_pretrained(model_checkpoints)
    model.push_to_hub(repo_id=IDENTIFIER,repo_type="model")

    # TOKENIZER
    tokenizer.save_pretrained(model_checkpoints)
    tokenizer.push_to_hub(repo_id=IDENTIFIER, repo_type="model")

    # MODEL CARD
    model_card.push_to_hub(IDENTIFIER)

#------------------------------------------------------------------------------#
# CLEAN UP BEFORE NEXT ITERATION
#------------------------------------------------------------------------------#

    del model, tokenizer, trainer
    torch.cuda.empty_cache()
    gc.collect()
```