ECE 60146 Assignment 10

Adhitha Dias

Wednesday 24th April, 2024; 18:14

Contents

1 Introduction

This document contains the results of the training and evaluation of the model on the SQuAD dataset. The model is trained on the SQuAD dataset using the DistilBERT model (for comparison) on question and answers, and locally fine-tuning a bert pretrained model. The model is trained for 100 epochs but the model overfits and the accuracy goes down at the 100th epoch. Therefore, I am reporting the results using the checkpoint obtained at the 35th epoch. All code snippets are borrowed from the class notes, Piazza posts, and the assignment description.

2 Output during training

```
{'loss': 2.484, 'grad_norm': 17.250259399414062, 'learning_rate': 4.714285714285714e-05, 'epoch': 0.57}
{'loss': 1.442, 'grad_norm': 17.010774612426758, 'learning_rate': 4.428571428571428e-05, 'epoch': 1.14}
{'loss': 0.9571, 'grad_norm': 39.44040298461914, 'learning_rate': 4.1428571428571437e-05, 'epoch': 1.71}
{'loss': 0.7126, 'grad_norm': 42.816986083984375, 'learning_rate': 3.857142857142858e-05, 'epoch': 2.29}
{'loss': 0.5209, 'grad_norm': 21.191295623779297, 'learning_rate': 3.57142857142857e-05, 'epoch': 2.86}
{'loss': 0.3601, 'grad_norm': 16.623889923095703, 'learning_rate': 3.285714285714286e-05, 'epoch': 3.43}
{'loss': 0.3125, 'grad_norm': 0.35003775358200073, 'learning_rate': 3e-05, 'epoch': 4.0}
{'loss': 0.2026, 'grad_norm': 0.2589515149593353, 'learning_rate': 2.714285714285714e-05, 'epoch': 4.57}
{'loss': 0.1959, 'grad_norm': 8.793052673339844, 'learning_rate': 2.4285714285714288e-05, 'epoch': 5.14}
{'loss': 0.1411, 'grad_norm': 52.41377639770508, 'learning_rate': 2.1428571428571428e-05, 'epoch': 5.71}
```

```
{'loss': 0.1168, 'grad_norm': 0.037031445652246475, 'learning_rate': 1.8571428571428572e-05, 'epoch': 6.29}
{'loss': 0.0986, 'grad_norm': 1.3157042264938354, 'learning_rate': 1.5714285714285715e-05, 'epoch': 6.86}
{'loss': 0.0565, 'grad_norm': 43.251949310302734, 'learning_rate': 1.2857142857142857e-05, 'epoch': 7.43}
{'loss': 0.062, 'grad_norm': 3.89984393119812, 'learning_rate': 1e-05, 'epoch': 8.0}
{'loss': 0.04, 'grad_norm': 0.0038578195963054895, 'learning_rate': 7.142857142857143e-06, 'epoch': 8.57}
{'loss': 0.0326, 'grad_norm': 0.0037382750306278467, 'learning_rate': 4.285714285714286e-06, 'epoch': 9.14}
{'loss': 0.0162, 'grad_norm': 0.0022480105981230736, 'learning_rate': 1.4285714285714286e-06, 'epoch':
9.71}
{'train_runtime': 12699.0025, 'train_samples_per_second': 5.512, 'train_steps_per_second': 0.689, 'train_loss': 0.4435298219953264, 'epoch': 10.0}
```

Listing 1: Output during the first 8 epochs of training

3 Qualitative Evaluation

Since the model returns uncased strings, I had to convert the expected answers to lowercase before comparing them. The evaluation script calculates the exact match and F1 score for each answer. The F1 score and exact match are calculated as shown in Listing8.

Because token join operation does not properly stitch the predicted answers, I have used several regular expression replacements to fix the predicted answers. The regular expressions used are shown in Listing6 and below.

```
    prediction = re.sub(r' ##', '', prediction)
    prediction = re.sub(r' ,', ',', prediction)
    prediction = re.sub(r' \.', '.', prediction)
    prediction = re.sub(r'\$ ', '$', prediction)
    prediction = re.sub(r' %', '%', prediction)
    prediction = re.sub(r' – ', '–', prediction)
    prediction = re.sub(r'\( ', '(', prediction)
    prediction = re.sub(r' \)', ')', prediction)
    prediction = re.sub(r' \' ', '\'', prediction)
    prediction = re.sub(r'. com', '.com', prediction)
    prediction = re.sub(ur' 0xE2 ', '0xE2', prediction)
    prediction = re.sub(r' 0', '0', prediction)
```

Listing 2: Regular Expressions used to fix the predicted answers

Note that these regular expressions do not fix the irregularities in the answers completely. I had to eye the output and select the regular expressions that I thought would fix the predicted answers.

Some of the examples of questions and answers are shown in Listing3. I have listed why the model did not get a perfect score in the comments.

```
—— the model is capable of selecting the correct answers with multiple words
Question: What are three games, in addition to Brick, which have been included with the iPod?
Predicted Answer: parachute, solitaire, and music quiz
Expected: parachute, solitaire, and music quiz
Exact match: 1
F1 score: 1.0
—— model is able to answer questions with numbers
Question: How many Chinese troops and medics were involved in the relief efforts?
Predicted Answer: 135,000
Expected: 135,000
Exact match: 1
F1 score: 1.0
—— model sometimes returns empty answers
Question: How tall is One World Trade Center in meters?
Predicted Answer:
Expected: 541.3
```

```
Exact match: 0
F1 score: 0
--- model return partial answers
Question: What was the first commercial solar concentrating system?
Predicted Answer: the solar total energy project
Expected: solar total energy project (step) in shenandoah, georgia, usa
Exact match: 0
F1 score: 0.5714285714285714
--- model answer is correct but not in the expected format, and I could not figure out how to fix it because xx, xx is a valid format for a comma separated list of values
Question: How many hectares does the island have in total?
Predicted Answer: 2, 500
Expected: 2,500
Exact match: 0
F1 score: 0
--- slight mismatch due to punctuation. There are several similar examples in the output that reduces the F1 score and exact match score
Question: What website showed video of an altercation between Mariah Carey and Nicki Minaj?
Predicted Answer: tmz.
Expected: tmz
Exact match: 0
F1 score: 0
--- right answer but slightly different letter in names, the tokenizer does not have this name in its vocabulary
Question: Who designed Chopin's tombstone?
Predicted Answer: clesinger
Expected: clésinger.
Exact match: 0
F1 score: 0
---
Question: What organization was Broca in the process of disentangling himself from?
Predicted Answer: societe de biologie
Expected: société de biologie
Exact match: 0
F1 score: 0.6666666666666666
--- sometimes does not include the article in the answer
Question: In what historical era does the book take place?
Predicted Answer: great depression
Expected: the great depression
Exact match: 0
F1 score: 0.8
--- model is able to answer questions with dates
Question: When was the Montana Territory formed?
Predicted Answer: april 26, 1864
Expected: april 26, 1864
Exact match: 1
F1 score: 1.0
--- just adding this here because I was born a Buddhist
Question: Who is the Buddha of this Buddha era?
Predicted Answer: gautama buddha
Expected: gautama buddha
Exact match: 1
F1 score: 1.0
--- model is able to answer questions with URLs or website names
Question: Where was the message posted?
Predicted Answer: cyberctm.com
Expected: cyberctm.com
Exact match: 1
F1 score: 1.0
---
```

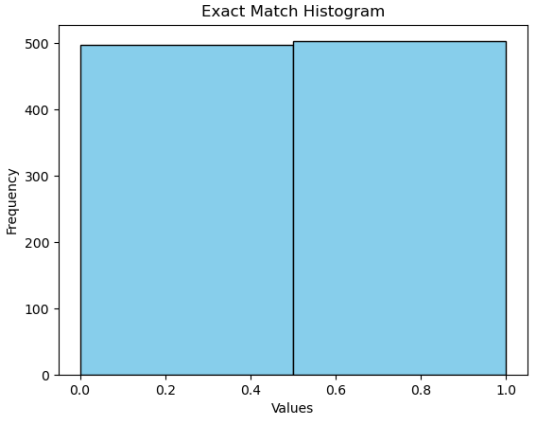
Listing 3: Example Question and Answer Outputs

4 Quantitative Evaluation with DistilBERT

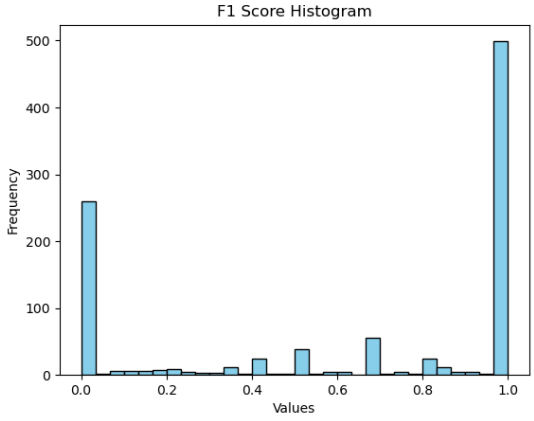
The model selects mostly correct answers as evident from the histograms in Figure1. The model has a mean exact match score of 0.769 and a median exact match score of 1.0. The mean F1 score is 0.891 and the median F1 score is 1.0. The comparison of the performance of the local model and DistilBERT is shown in Table1.

	Mean Exact Match	Median Exact Match	Mean F1 Score	Median F1 Score
Local	0.509	1.0	0.636	1.0
DistilBERT	0.769	1.0	0.891	1.0

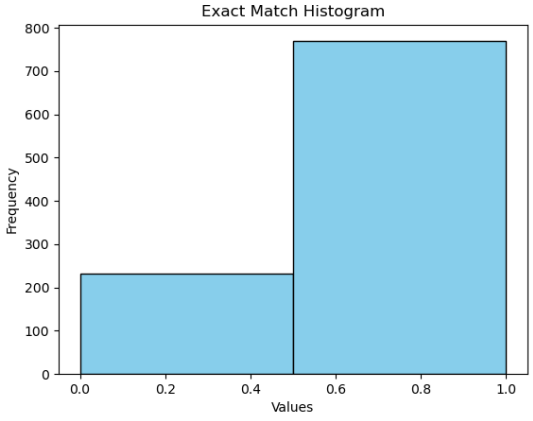
Table 1: Comparison of the performance of the local model and DistilBERT



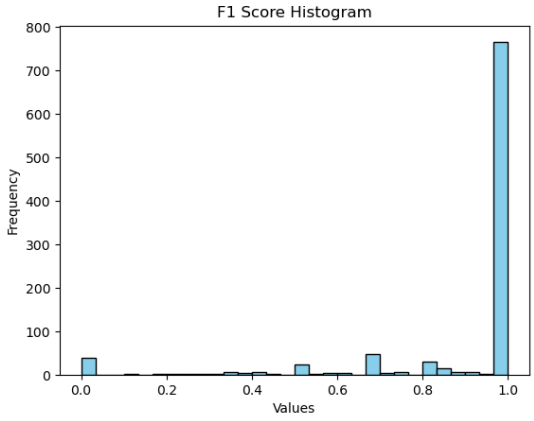
(a) Histogram of Exact Match Scores



(b) Histogram of F1 Scores



(c) Histogram of Exact Match Scores for DistilBERT



(d) Histogram of F1 Scores for DistilBERT

Figure 1: Histograms of Exact Match and F1 Scores

5 Code Snippets

```python
import pickle
import pandas as pd
from datasets import Dataset

with open('dataset/train_dict.pkl', 'rb') as f:
    train_dict = pickle.load(f)

with open('dataset/test_dict.pkl', 'rb') as f:
    test_dict = pickle.load(f)

with open('dataset/eval_dict.pkl', 'rb') as f:
    eval_dict = pickle.load(f)

with open('dataset/train_data_processed.pkl', 'rb') as f:
    train_processed = pickle.load(f)

with open('dataset/test_data_processed.pkl', 'rb') as f:
    test_processed = pickle.load(f)

with open('dataset/eval_data_processed.pkl', 'rb') as f:
    eval_processed = pickle.load(f)

print(train_dict.keys())
print(test_dict.keys())
print(eval_dict.keys())

print(train_processed.keys())
print(test_processed.keys())
print(eval_processed.keys())

# print one element of the train_dict
for k,v in train_dict.items():
    print('key', k)
    print('value', len(v))
    # break

train_dataset = Dataset.from_pandas(pd.DataFrame(train_processed))
test_dataset = Dataset.from_pandas(pd.DataFrame(test_processed))
eval_dataset = Dataset.from_pandas(pd.DataFrame(eval_processed))
```

Listing 4: Dataset Definition

```python
from src.dataset import \
    train_dict, test_dict, eval_dict, \
    train_processed, test_processed, eval_processed, \
    train_dataset, test_dataset, eval_dataset

from transformers import BertForQuestionAnswering

model_name = 'bert-base-uncased'
model = BertForQuestionAnswering.from_pretrained(model_name)

# load the model from the saved checkpoint
# model = BertForQuestionAnswering.from_pretrained('./results/checkpoint-2000')

print(model._modules)

from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir='./results',          # output directory
    use_mps_device=False,            # use 16-bit (mixed precision) training
    num_train_epochs=100,              # number of training epochs
    per_device_train_batch_size=8,   # batch size for training
    per_device_eval_batch_size=8,    # batch size for evaluation
    warmup_steps=0,                  # number of warmup steps for learning rate scheduler
    weight_decay=0.01,               # strength of weight decay
    logging_dir='./logs',            # directory for storing logs
```

```
    # save_steps=1_000,               # frequency of model checkpoints
)

from transformers import Trainer

trainer = Trainer(
    model=model,                         # the instantiated Transformers model to be trained
    args=training_args,                  # training arguments, defined above
    train_dataset=train_dataset,         # training dataset
    eval_dataset=eval_dataset            # evaluation dataset
)

print('training the network')
trainer.train()
```

Listing 5: Training Script

```
from src.dataset import \
    train_dict, test_dict, eval_dict, \
    train_processed, test_processed, eval_processed, \
    train_dataset, test_dataset, eval_dataset

from transformers import BertForQuestionAnswering

model_name = 'bert-base-uncased'
# model = BertForQuestionAnswering.from_pretrained(model_name)

# load the model from the saved checkpoint
model = BertForQuestionAnswering.from_pretrained('./results/checkpoint-31500')

print(model._modules)

from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir='./results',          # output directory
    use_mps_device=False,            # use 16-bit (mixed precision) training
    num_train_epochs=10,              # number of training epochs
    per_device_train_batch_size=8,   # batch size for training
    per_device_eval_batch_size=8,    # batch size for evaluation
    warmup_steps=0,                  # number of warmup steps for learning rate scheduler
    weight_decay=0.01,               # strength of weight decay
    logging_dir='./logs',            # directory for storing logs
    # save_steps=1_000,               # frequency of model checkpoints
)

from transformers import Trainer

trainer = Trainer(
    model=model,                         # the instantiated Transformers model to be trained
    args=training_args,                  # training arguments, defined above
    train_dataset=train_dataset,         # training dataset
    eval_dataset=eval_dataset            # evaluation dataset
)

print('NOT training the network')
# trainer.train()

import numpy as np
from transformers import BertTokenizer

# Initialize the tokenizer
tokenizer = BertTokenizer.from_pretrained(model_name)
# tokenizer = BertTokenizer.from_pretrained('./results/checkpoint-2000')


x = trainer.predict(test_dataset)
start_pos, end_pos = x.predictions
start_pos = np.argmax(start_pos, axis=1)
end_pos = np.argmax(end_pos, axis=1)

```

```
from src.util import compute_exact_match, f1_score

lexact_match = []
lf1_score = []

import re

for k, (i, j) in enumerate(zip(start_pos, end_pos)):
    tokens = tokenizer.convert_ids_to_tokens(test_processed['input_ids'][k])
    print('Question:', test_dict['question'][k])

    prediction = ' '.join(tokens[i:j+1])
    prediction = re.sub(r' ##', '', prediction)
    prediction = re.sub(r' ,', ',', prediction)
    prediction = re.sub(r' \.', '.', prediction)
    prediction = re.sub(r'\$ ', '$', prediction)
    prediction = re.sub(r' %', '%', prediction)
    prediction = re.sub(r' - ', '-', prediction)
    prediction = re.sub(r'\( ', '(', prediction)
    prediction = re.sub(r' \)', ')', prediction)
    prediction = re.sub(r' \' ', '\'', prediction)
    prediction = re.sub(r'. com', '.com', prediction)
    prediction = re.sub(r' \ ', '', prediction)
    prediction = re.sub(r' 0', '0', prediction)
    print('Predicted Answer:', prediction)
    ground_truth = test_dict['answers'][k]['text'][0]
    ground_truth = ground_truth.lower()
    print('Expected:', ground_truth)

    exmatch = compute_exact_match(prediction, ground_truth)
    f1 = f1_score(prediction, ground_truth)
    print('Exact match:', exmatch)
    print('F1 score:', f1)

    lexact_match.append(exmatch)
    lf1_score.append(f1)

    print('---')

from src.util import print_stats, plot_histogram

print_stats(lexact_match, lf1_score)
plot_histogram(lexact_match, 2, 'Values', 'Frequency', 'Exact Match Histogram', 'images/
    histogram_exact_match.png')
plot_histogram(lf1_score, 30, 'Values', 'Frequency', 'F1 Score Histogram', 'images/
    histogram_f1_score.png')
```

Listing 6: Evaluation Script

```
from src.util import compute_exact_match, f1_score
from src.dataset import \
    train_dict, test_dict, eval_dict, \
    train_processed, test_processed, eval_processed, \
    train_dataset, test_dataset, eval_dataset

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

from transformers import pipeline

question_answerer = pipeline('question-answering', model='distilbert-base-cased-distilled-squad')

lexact_match = []
lf1_score = []

for i in range(len(test_dict['question'])):
    result = question_answerer(context=test_dict['context'][i], question=test_dict['question'][i])

    question = test_dict['question'][i]
    print('Question:', question)
```

```
    answer = result['answer']
    print('Answer:', answer)
    correct_answer = test_dict['answers'][i]['text'][0]
    print('Correct Answer:', correct_answer)
    exmatch = compute_exact_match(answer, correct_answer)
    print('Exact Match:', compute_exact_match(answer, correct_answer))
    f1 = f1_score(answer, correct_answer)
    print('F1 Score:', f1)
    lexact_match.append(exmatch)
    lf1_score.append(f1)
    print('---')

from src.util import print_stats
from src.util import plot_histogram

print_stats(lexact_match, lf1_score)
plot_histogram(lexact_match, 2, 'Values', 'Frequency', 'Exact Match Histogram', 'images/
    histogram_comp_exact_match.png')
plot_histogram(lf1_score, 30, 'Values', 'Frequency', 'F1 Score Histogram', 'images/
    histogram_comp_f1_score.png')
```

Listing 7: Script to run distilbert-base-cased-distilled-squad

```
def compute_exact_match(prediction, truth):
    return int(prediction == truth)

def f1_score(prediction, truth):
    prediction_tokens = prediction.split()
    truth_tokens = truth.split()

    # if either the prediction or the truth is no-answer then f1 = 1 if they agree, 0 otherwise
    if len(prediction_tokens) == 0 or len(truth_tokens) == 0:
        return int(prediction_tokens == truth_tokens)

    common_tokens = set(prediction_tokens) & set(truth_tokens)

    # if there are no common tokens then f1 = 0
    if len(common_tokens) == 0:
        return 0

    precision = len(common_tokens) / len(prediction_tokens)
    recall = len(common_tokens) / len(truth_tokens)
    if precision + recall == 0:
        return 0

    return 2 * (precision * recall) / (precision + recall)

def plot_histogram(l, bins, xlabel, ylabel, title, filename):
    import matplotlib.pyplot as plt
    plt.hist(l, bins=bins, color='skyblue', edgecolor='black')
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.savefig(filename)
    plt.clf()

def print_stats(lexact_match, lf1_score):
    import numpy as np
    print(lexact_match)
    print(lf1_score)
    mean_exact_match = np.mean(lexact_match)
    mean_f1_score = np.mean(lf1_score)

    median_exact_match = np.median(lexact_match)
    median_f1_score = np.median(lf1_score)

    print('Mean Exact Match:', mean_exact_match)
    print('Mean F1 Score:', mean_f1_score)

    print('Median Exact Match:', median_exact_match)
```

```
48     print('Median F1 Score:', median_f1_score)
```

Listing 8: Utility Functions

References

[1] Class Notes

[2] DLStudio, https://engineering.purdue.edu/kak/distDLS/