1_0_embedding_model_finetuning (1)

October 23, 2024

[ ]:
```
# install all the necessary packages
! pip install datasets langchain_community sentence_transformers chromadb␣
 ↪langchain_openai ragas==0.0.19 --q
! pip install transformers
! pip install accelerate
```

[12]:
```
# Importing the required libraries

import os
from tqdm import tqdm
import pandas as pd
from datasets import Dataset


# Importing the required libraries

HF_TOKEN= "{HF_TOKEN}"
OPENAI_API_KEY = "{OPENAI_API_KEY}"
WANDB_API_KEY = "{WANDB_API_KEY}"
```

[ ]:
```
from datasets import load_dataset

# Load dataset from the hub  and rename the columns
train_dataset = load_dataset("neural-bridge/rag-dataset-12000", split="train")
#  Rename the columns
train_dataset = train_dataset.rename_column("question", "anchor")
# Rename the columns
train_dataset = train_dataset.rename_column("context", "positive")
# Rename the columns
train_dataset = train_dataset.remove_columns('answer')

df = train_dataset.to_pandas()
print(df.isnull().sum())
df.dropna(inplace=True)
print(df.isnull().sum())
train_dataset  = Dataset.from_pandas(df)
```

[ ]:
```
# Remove the index column
train_dataset = train_dataset.remove_columns('__index_level_0__')
train_dataset # Display the dataset
```

[ ]:
```
Dataset({
    features: ['positive', 'anchor'],
    num_rows: 9598
})
```

[ ]:
```
# Load the test dataset and rename the columns
test = pd.read_csv('/content/drive/MyDrive/bge-base-en-v1.
 ↪5-finetuned_osllmai_v1/ragdataset/sampled_500_test_set.csv')
test = test.loc[:,['context','question']].rename(columns={'context':␣
 ↪'positive', 'question': 'anchor'})
eval_dataset  = Dataset.from_pandas(test)
```

[ ]:
```
# Remove the index column
print(train_dataset) , print(eval_dataset)
```

```
Dataset({
    features: ['positive', 'anchor'],
    num_rows: 9598
})
Dataset({
    features: ['positive', 'anchor'],
    num_rows: 500
})
```

[ ]:
```
(None, None)
```

[ ]:
```
!nvidia-smi # Display the GPU information
```

```
Tue Oct 22 02:36:16 2024
+---------------------------------------------------------------------------------
--------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version:
12.2     |
|-----------------------------------------+----------------------+-------------
--------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile
Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util
Compute M. |
|                                         |                      |
MIG M. |
|=========================================+======================+=============
========|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |
```

```
0 |
| N/A   46C    P8               9W /  70W |      0MiB / 15360MiB |      0%
Default |
|                                         |                       |
N/A |
+-----------------------------------------+-----------------------+-------------
--------+

+---------------------------------------------------------------------------------
--------+
| Processes:
|
|  GPU   GI   CI        PID   Type   Process name                             GPU
Memory |
|        ID   ID
Usage      |
|=================================================================================
========|
|  No running processes found
|
+---------------------------------------------------------------------------------
--------+
```

[ ]:
```
from sentence_transformers import SentenceTransformer
import torch
# https://huggingface.co/BAAI/bge-base-en-v1.5
model_name = "BAAI/bge-base-en-v1.5" # Load the model
model = SentenceTransformer(model_name)
```

[ ]:
```
from sentence_transformers import SentenceTransformerTrainer,␣
↪SentenceTransformerTrainingArguments
from sentence_transformers.losses import MultipleNegativesRankingLoss
from sentence_transformers.training_args import BatchSamplers
```

[ ]:
```
# 3. Define a loss function which will be used to train the model
loss = MultipleNegativesRankingLoss(model)
```

[ ]:
```
args = SentenceTransformerTrainingArguments(
    # Required parameter: define the filename where the model will be saved
    output_dir="/content/drive/MyDrive/bge-base-en-v1.5-finetuned_osllmai_v1/
↪models/bge-base-en-v1.5-ft_ragds",
    # Optional training parameters: batch size, maximum number of epochs
    num_train_epochs=30, # We train 3 epochs, this is just an example
    per_device_train_batch_size=10, # Number of samples per batch
    per_device_eval_batch_size=10, # Number of samples per batch
    warmup_ratio=0.1, # Warmup 10% of the training time
```

```
    fp16=False,  # Set to False if you get an error that your GPU can't run on␣
 ↪FP16
    bf16=True,  # Set to True if you have a GPU that supports BF16
    batch_sampler=BatchSamplers.NO_DUPLICATES,  # MultipleNegativesRankingLoss␣
 ↪benefits from no duplicate samples in a batch
    # Optional tracking/debugging parameters:
    eval_strategy="steps", # Evaluate model after each epoch
    eval_steps=100, # Number of update steps between two evaluations
    save_strategy="steps", # Save the model after each epoch
    save_steps=1200, # Number of updates steps before two checkpoints
    save_total_limit=1, # Number of maximum checkpoints that are saved
    logging_steps=500, # Log every 500 steps
    run_name="bge-base-en-v1.5-finetuned_ragds_v1",  # Will be used in W&B if␣
 ↪`wandb` is installed
    eary
)
```

[ ]:
```
# The active selection creates an instance of SentenceTransformerTrainer
# with the following parameters:

# model: The sentence transformer model to fine-tune.
# args: Training hyperparameters.
# train_dataset: Dataset for training.
# eval_dataset: Dataset for evaluation.
# loss: Loss function for optimization.

trainer = SentenceTransformerTrainer(
    model=model,
    args=args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    loss=loss
)
```

[ ]:
```
trainer.train() # Train the model
```

```
wandb: Using wandb-core as the SDK backend. Please refer to
https://wandb.me/wandb-core for more information.

<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: ERROR API key must be 40 characters long,
yours was 38
```

```
<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
```

```
<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>
```

```
WARNING:sentence_transformers.data_collator:Column 'anchor' is at index 1,
whereas a column with this name is usually expected at index 0. Note that the
column order can be important for some losses, e.g. MultipleNegativesRankingLoss
will always consider the first column as the anchor and the second as the
positive, regardless of the dataset column names. Consider renaming the columns
to match the expected order, e.g.:
dataset = dataset.select_columns(['anchor', 'positive', 'negative'])
```

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

```
Computing widget examples:   0%|          | 0/1 [00:00<?, ?example/s]
```

1 Load the model

[5]:
```
%ls /content/drive/MyDrive/bge-base-en-v1.5-finetuned_osllmai_v1/models/
  ↪bge-base-en-v1.5-ft_ragds
```

```
checkpoint-7900/  checkpoint-8000/
checkpoint-8100/  runs/
```

[9]:
```
from sentence_transformers import SentenceTransformer

# Path to the directory where the model weights were saved
checkpoint_dir = "/content/drive/MyDrive/bge-base-en-v1.5-finetuned_osllmai_v1/
  ↪models/bge-base-en-v1.5-ft_ragds/checkpoint-7900"

# Load the model from the checkpoint
model = SentenceTransformer(checkpoint_dir)

from sentence_transformers import SentenceTransformer

# Run inference
```

```
sentences = [
    'Early this year, there was a buzz on Motorola inviting T-Mobile Moto X␣
 ↪owners to take part in a soak test for possible future update. Motorola␣
 ↪seemed skeptical in disclosing facts at that point of time but since Moto G␣
 ↪was recently upgraded to Android 4.4.2; enthusiasts anticipated the same for␣
 ↪T-Mobile Moto X. And it turned out to be true.\nNews Update\nThis T-Mobile␣
 ↪version of Moto X is now receiving the upgrade which is a file size of 147.6␣
 ↪MB. The Android 4.4.2 is the latest version of KitKat that includes all the␣
 ↪goodies from the earlier installments, plus a few additions. The good news␣
 ↪is, Motorola has customized the whole package and made a few tweaks into the␣
 ↪update. The Software Version bumped to 161.44.25 and the notable changes are␣
 ↪listed as below:\n- It added substantial support for services like printing␣
 ↪photos, Google Docs, Gmail messages and other such content via Wi-Fi,␣
 ↪Bluetooth and hosted services such as HP ePrinters and Google Cloud Print.
 ↪\n- It fixed all the bugs identified during the preliminary runs, including␣
 ↪the ones that caused a few users to experience short battery life after␣
 ↪upgrading to KitKat.\n- Another bug that caused delays in synchronizing␣
 ↪email services like Microsoft Exchange was resolved, thus adding to the␣
 ↪convenience of the user.\nThis is a noteworthy upgrade, considering the fact␣
 ↪that bugs and errors were fixed. Mobile addicts across the world will␣
 ↪rejoice, for they can experience the smartness of Android KitKat flawlessly␣
 ↪in their devices. This is significant development in terms of update.\nThis␣
 ↪variant is an unlocked GSM device so chances are, you can use it on networks␣
 ↪of other service providers. In all probability, the update should not be␣
 ↪affected and the installation should hardly take much time. The T-Mobile␣
 ↪Moto X Android update is now available for manual download. It is accessible␣
 ↪in the following sequential way:\n- Click on Settings\n- Click on About␣
 ↪Phone\n- Click on System Updates\n- Click on Download\nRecommendations\nFor␣
 ↪ensuring a successful installation, it is highly recommended to install this␣
 ↪update with at least 50% battery and a strong connectivity; preferably Wi-Fi.
 ↪ Follow the notification message and select download-> once the download is␣
 ↪over, select Install-> Once the installation is over, and the phone will␣
 ↪automatically restart. This marks the completion of the installation process.
 ↪ The phone is now updated to 161.44.25 - This build is same as the soak test.
 ↪\nThis upgrade is free in the carrier network and Motorola and Google has␣
 ↪collaborated for a back up service for those in trouble. In case of␣
 ↪distress, a user can contact them through the Moto X web interface and avail␣
 ↪the service. There is still no news on other carrier variants of this update␣
 ↪but we can safely hope that it will roll out very soon. Though the upgrade␣
 ↪doesn't appeal in terms of version number but it is definitely significant␣
 ↪for users to live with the latest KitKat.',
    'What are some of the notable changes in the T-Mobile Moto X update?',
    'Who is the editor of "The Routledge Handbook of Tourism Geographies"?',
]
embeddings = model.encode(sentences)
print(embeddings.shape)
# [3, 768]
```

```
# Get the similarity scores for the embeddings
similarities = model.similarity(embeddings, embeddings)
print(similarities.shape)
# [3, 3]
```

```
(3, 768)
torch.Size([3, 3])
```

[10]:
```
from sentence_transformers import SentenceTransformer

# Run inference
sentences = [
    'The latest version of Tesla's Full Self-Driving (FSD) beta software, v11, introduces a more unified architecture for autopilot functionality. In addition to standard improvements like enhanced lane keeping and better recognition of traffic signs, the FSD v11 update now supports improved decision-making in unprotected left turns and smoother highway merges.\nNews Update\nTesla has also added a new energy app to track energy consumption and efficiency while driving. Moreover, improvements to cabin and exterior camera systems have increased their range of vision, offering better nighttime performance. However, some users are reporting sporadic glitches with the autopilot disengaging at certain highway exits. Tesla is expected to address these issues in the next release.\nThis update is accessible via the car's over-the-air (OTA) system, and users can expect the installation process to take around 30-40 minutes depending on their current software version.\n- Click on the Tesla "T" logo in the app\n- Go to the Software section\n- Click on the Update button\nRecommendations\nEnsure the car is plugged in and charging during the update installation. Tesla recommends doing this at home for the most stable connection to their servers. If an error occurs during the update, reach out to Tesla Support for further assistance. Many users are excited for what's next, as Elon Musk hinted at even more significant enhancements in future versions, including potential features for a robotaxi fleet.',
    'What is the latest version of Tesla's Full Self-Driving software?',
    'What are some of the features improved in FSD v11?',
    'What are users experiencing after the update?',
]
embeddings = model.encode(sentences)
print(embeddings.shape)
# [4, 768]

# Get the similarity scores for the embeddings
similarities = model.similarity(embeddings, embeddings)
print(similarities.shape)
# [4, 4]
```

```
(4, 768)
```

```
torch.Size([4, 4])
```

Let me explain what each part of the code is doing:

1.0.1 1. Embeddings Calculation:

```
embeddings = model.encode(sentences)
print(embeddings.shape)
# [4, 768]
```

- `model.encode(sentences)`:
  - The `SentenceTransformer` model is used to convert the list of sentences into numerical vectors, called *embeddings*. Each sentence is represented as a high-dimensional vector in a way that captures the meaning of the sentence.
  - Here, the model takes a list of 4 sentences (`sentences`) and encodes them into their corresponding embeddings (numerical representations).
- **What are Embeddings?**:
  - An embedding is a dense vector representation of the text. These embeddings are designed to capture the semantic meaning of the sentences, so similar sentences (in meaning) will have embeddings that are close together in the vector space.
- **embeddings.shape**:
  - The output of `embeddings.shape` tells us the dimensions of the resulting tensor (a multi-dimensional array).
  - The result `[4, 768]` means:
    - `4` refers to the number of sentences provided (4 sentences).
    - `768` refers to the size (or dimensionality) of each embedding vector.
  - So each sentence is represented by a 768-dimensional vector.

1.0.2 2. Similarity Scores Calculation:

```
similarities = model.similarity(embeddings, embeddings)
print(similarities.shape)
# [4, 4]
```

- `model.similarity(embeddings, embeddings)`:
  - This calculates the pairwise similarity between the embeddings of the sentences.
  - Similarity scores indicate how close the meanings of two sentences are in the embedding space.
  - The function compares each sentence embedding to all other sentence embeddings (including itself) to generate a similarity score, usually using a metric like cosine similarity.
- **similarities.shape**:
  - The output `[4, 4]` is the shape of the similarity matrix.
  - This is a 4x4 matrix because you have 4 sentences, and you're comparing each sentence to the other 3 (and to itself).
  - In this matrix:
    - The diagonal elements (i.e., comparing a sentence to itself) will typically have the highest score (often close to 1).
    - Off-diagonal elements (comparing different sentences) will show the similarity between those different sentences.

Example of how the similarity matrix works: If the sentences were: - Sentence 1 - Sentence 2 - Sentence 3 - Sentence 4

The similarity matrix could look something like this:

```
[[1.0, 0.8, 0.5, 0.3],
 [0.8, 1.0, 0.4, 0.2],
 [0.5, 0.4, 1.0, 0.7],
 [0.3, 0.2, 0.7, 1.0]]
```

- The diagonal values (e.g., comparing Sentence 1 to itself) are 1.
- The off-diagonal values (e.g., comparing Sentence 1 to Sentence 2) represent the similarity score between those sentences.

1.0.3 Summary:

1. **Embedding shape `[4, 768]`**: 4 sentences were encoded into 768-dimensional embeddings.
2. **Similarity matrix shape `[4, 4]`**: A 4x4 similarity matrix was created to represent the similarity scores between all pairs of 4 sentences.

1.0.4 Interpretation of the Training and Validation Loss

1. **Training Loss**: This measures how well the model is performing on the training data. A lower training loss means the model is fitting the training data well. In your results, the training loss starts high and gradually decreases, indicating the model is learning over time.
2. **Validation Loss**: This measures how well the model generalizes to unseen data (validation data). Ideally, you want this to also decrease over time. If validation loss starts increasing while training loss keeps decreasing, the model might be overfitting (fitting too closely to the training data and not generalizing well).

1.0.5 Key Observations:

- **Initial Improvement (Steps 100 to 800)**: Both training and validation loss decrease consistently, showing that the model is improving its performance on both training and validation data. Validation loss goes from 0.052656 at step 100 to 0.011138 at step 800, showing good progress.
- **Best Performance (Around Step 1000)**: The validation loss reaches its lowest point around step 1000 with a value of 0.010798. This indicates the model is performing best around this point.
- **Fluctuations and Overfitting Signs (After Step 1000)**:
  - After step 1000, validation loss starts fluctuating and even increases in some places, indicating possible overfitting. For example, at step 2000, validation loss jumps to 0.016175, and by step 4000, it hits 0.018775.
  - The training loss, however, continues to decrease, indicating the model is continuing to fit the training data better but not improving on the validation data.
- **Overfitting (After Step 3000)**: After step 3000, the validation loss increases significantly (reaching 0.024351 and higher), while the training loss remains very low. This is a clear sign

of overfitting. The model has learned the training data too well and is not generalizing well to the validation set.

1.0.6 Best Model:

The best model is likely at or near **step 1000**, where the **validation loss is lowest** (0.010798) before any significant signs of overfitting. This is the point where the model's generalization to unseen data is best.

1.0.7 Recommendation:

- **Stop Training Around Step 1000**: The model performs best at this point, with the lowest validation loss. Continuing training past this point introduces overfitting.
- **Early Stopping**: Consider using early stopping in future training runs to automatically halt training when the validation loss stops improving to avoid overfitting.