Milestone 3

Scott Chase Waggener
Pankaj Kr Jhawar
Yuan Chiang

November 20, 2019

1 Baseline System

1.1

We implemented BertSum as our baseline, which was published on two occasions in 2019 to arXiv. BertSum used a pretrained BERT model and clever preprocessing to produce sentence level embeddings, which are attached to the [CLS] start of sentence tokens. These sentence level embeddings were then summed with positional encodings (Vaswani et al.) and sentence segment embeddings. Finally, a two level encoder transformer was used to map these embeddings to strong features, which were subsequently mapped to a probability via a dense + sigmoid layer. The output probabilities represent the value of including a sentence in the summary.

1.2

The model checkpoint file is available at the following link. The checkpoint file is $1.5GB$ in size.

`https://drive.google.com/open?id=1gEXSIOkxwAK_H1gE95OMXmoWI6Y2VwUg`

The training/testing pipeline is Dockerized for ease of use. Use of this model requires several Python libraries (installable from pip) which are listed

in the REQUIREMENTS.TXT file. The Perl ROUGE-1.5.5 script is also required.

Pytorch was used for the implementation. While we initially planned to use Tensorflow, many of BertSum's implementation details were more subtle than what Tensorflow could easily accomodate. The OpenNMT library was used to facilitate things like multi-GPU computation and logging. PYTORCH-PRETRAINED-BERT was used to obtain a pretrained BERT model. The published BertSum implementation was used as a reference, though large portions have been re-implemented from scratch using Pytorch builtin functions.

Relevant Docker and Docker Compose files are provided to make training/testing simpler. The scripts TRAIN.SH and TEST.SH are provided that invoke docker compose with good defaults for training and testing. When using docker compose, the .ENV file is read to determine how to mount the input dataset and runtime artifacts on the docker container. Override the contents of .ENV as desired.

Sample execution using docker compose to evaluate the model at checkpoint step 50000 on GPU (this will take around 12 minutes).

```
docker-compose run bertsum  \
--mode test  \
--test_from /artifacts/models/model_step_50000.pt  \
--encoder bertsum  \
--visible_gpus 1  \
--gpu_ranks 0  \
--batch_size 4
```

An example execution outside of Docker (The pipeline has not been tested outside of Docker.)

```
docker-compose run bertsum  \
--mode test  \
--test_from /artifacts/models/model_step_50000.pt  \
--encoder bertsum  \
--visible_gpus 1  \
--gpu_ranks 0  \
--batch_size 4
```

The full help output of TRAIN.PY, which runs evaluation when given the –MODE="TEST" flag.

```
docker-compose run bertsum  \
--mode test  \
--test_from /artifacts/models/model_step_50000.pt  \
--encoder bertsum  \
--visible_gpus 1  \
--gpu_ranks 0  \
--batch_size 4
```

2

2.1

Our training process falls somewhere between supervised and semi-supervised learning. The labels for our data were obtained using a greedy algorithm that generates sentences that maximize ROUGE-2 against the gold standard summary. The model had many hyperparameters, including the number of transformer layers, dropout fraction, and the standard training-time hyperparameters. Due to the computational cost of training this model we did not spend a great deal of time tuning hyperparameters, though we plan to rectify this for the final milestone. Our choice of train-time parameters was informed by those given in the BertSum paper, adjusted to account for differences in batch size. Training completed overnight on two GTX970 GPUs. These GPUs have significantly less memory than the 3x GTX1080Ti GPUs used in the BertSum paper, meaning that we were constrained to small batch sizes.

3

3.1

The EVALUATE.PY file is just a wrapper over TRAIN.PY. Sample usage via docker compose is as follows

First, set your dataset and artifact mount points in the .ENV file.

```
SRC_DIR=/mnt/iscsi/cnn_dailymail # Mounted at /data
ARTIFACT_DIR=/mnt/iscsi/bertsum # Mounted at /artifacts
```

In our case, the artifacts directory had the following subdirectories

```
logs
models
results
temp
```

Next, start evaluation by running EVALUATE.PY via docker compose.

```
docker-compose run \
--entrypoint 'python /app/evaluate.py' \
bertsum \
--test_from /artifacts/models/model_step_50000.pt  \
--encoder bertsum \
--dropout 0.1 \
--learning_rate 1e-3 \
--visible_gpus 1 \
--gpu_ranks 0 \
--world_size 1 \
--report_every 50 \
--save_checkpoint_steps 1000 \
--batch_size 1 \
--decay_method noam \
--train_steps 50000 \
--accum_count 2 \
--heads 8 \
--use_interval true \
--warmup_steps 10000
```

Some of these parameters are not necessary or will differ depending on whether or not the test environment is GPU equipped. Note that the submission contains TRAIN.SH and TEST.SH scripts that demonstrate docker compose usage to train and evaluate respectively.

3.2

ROUGE 1, 2, and L were used as evaluation metrics. Rouge 1 and 2 examine the overlap of unigrams and bigrams respectively, while rouge L identifies the longest common subsequence. These metrics are widely used for summarization, and convey a measure of overlap and fluency. Our results are shown below, as evaluated on the last checkpoint. Our implementation was less performant than the author's cited results, which is likely a result of less optimal hyperparameter tuning and training schedule.

For example, we estimated a reduced learning rate that would account for our lower batch sizes. We also trained for the same number of steps as the authors, meaning that our model may not have yet converged. Finally, we did not perform the costly best model selection process that the author used, as this would require evaluation of all 50 checkpointed models.

```
------------------------------------------
1 ROUGE-1 Average_R: 0.46877 (95%-conf.int. 0.46382 - 0.47352)
1 ROUGE-1 Average_P: 0.25893 (95%-conf.int. 0.25533 - 0.26248)
1 ROUGE-1 Average_F: 0.32246 (95%-conf.int. 0.31867 - 0.32631)
------------------------------------------
1 ROUGE-2 Average_R: 0.24886 (95%-conf.int. 0.24395 - 0.25379)
1 ROUGE-2 Average_P: 0.13747 (95%-conf.int. 0.13400 - 0.14089)
1 ROUGE-2 Average_F: 0.17058 (95%-conf.int. 0.16670 - 0.17441)
------------------------------------------
1 ROUGE-3 Average_R: 0.15347 (95%-conf.int. 0.14868 - 0.15812)
1 ROUGE-3 Average_P: 0.08513 (95%-conf.int. 0.08188 - 0.08833)
1 ROUGE-3 Average_F: 0.10493 (95%-conf.int. 0.10139 - 0.10844)
------------------------------------------
1 ROUGE-4 Average_R: 0.10866 (95%-conf.int. 0.10421 - 0.11281)
1 ROUGE-4 Average_P: 0.06076 (95%-conf.int. 0.05783 - 0.06375)
1 ROUGE-4 Average_F: 0.07428 (95%-conf.int. 0.07096 - 0.07756)
------------------------------------------
1 ROUGE-L Average_R: 0.41217 (95%-conf.int. 0.40712 - 0.41706)
1 ROUGE-L Average_P: 0.22885 (95%-conf.int. 0.22528 - 0.23236)
1 ROUGE-L Average_F: 0.28447 (95%-conf.int. 0.28045 - 0.28838)
------------------------------------------
1 ROUGE-W-1.2 Average_R: 0.22583 (95%-conf.int. 0.22297 - 0.22858)
```

```
1 ROUGE-W-1.2 Average_P: 0.20933 (95%-conf.int. 0.20593 - 0.21274)
1 ROUGE-W-1.2 Average_F: 0.20818 (95%-conf.int. 0.20535 - 0.21099)
---------------------------------------------
1 ROUGE-S* Average_R: 0.25424 (95%-conf.int. 0.24890 - 0.25950)
1 ROUGE-S* Average_P: 0.08691 (95%-conf.int. 0.08381 - 0.08993)
1 ROUGE-S* Average_F: 0.11560 (95%-conf.int. 0.11222 - 0.11884)
---------------------------------------------
1 ROUGE-SU* Average_R: 0.28325 (95%-conf.int. 0.27801 - 0.28842)
1 ROUGE-SU* Average_P: 0.09991 (95%-conf.int. 0.09679 - 0.10293)
1 ROUGE-SU* Average_F: 0.13283 (95%-conf.int. 0.12940 - 0.13612)

ROUGE-F(1/2/3/l): 32.25/17.06/28.45
ROUGE-R(1/2/3/l): 46.88/24.89/41.22

[2019-11-20 00:58:48,334 INFO] Rouges at step 30000
ROUGE-F(1/2/3/l): 32.25/17.06/28.45
ROUGE-R(1/2/3/l): 46.88/24.89/41.22
```