Binary classification with pre-trained BERT

Yifan (Rosetta) Hu
Loyola Marymount University
yhu9@lion.lmu.edu

Mandy Korpusik, PhD
Loyola Marymount University
mandy.korpusik@lmu.edu

December 19, 2022

1 Team

Undergraduate student Rosetta Hu was responsible for writing the Python script that preprocesses the 2015-2016 UPC, EC, and PPC data fortraining neural network models. Her script randomly sampled five negative EC descriptions for every postive match between a UPC and EC code. Professor Korpusik performed the remaining work, including setting up the environment, training the BERT model, and evaluation. Rosetta spent roughly 10 hours on the competition, and Dr. Korpusik spent roughly 40 hours of work (and many additional hours running and monitoring the training and testing scripts).

2 Summary

The goal of this challenge is to use machine learning and natural language processing to link language-based entries in the IRI and FNDDS databases. Our proposed approach is based on our prior work using deep learning models to map users' natural language meal descriptions to the FNDDS database to retrieve nutrition information in a spoken diet tracking system. In the past, we found a tradeoff between accuracy and cost, leading us to select convolutional neural networks over recurrent LSTMs. Here, we propose to investigate state-of-the-art Transformers, specifically the contextual embedding model (i.e., the entire sentence is used as context to generate the embedding) known as BERT (Bidirectional Encoder Representations from Transformers) [1].

Our approach is to fine-tune a large pre-trained BERT language model on the food data. BERT was originally trained on a massive amount of text for a language modeling task (i.e., predicting which word should come next in a sentence). It relies on a Transformer model, which uses an "attention" mechanism to identify which words the model should pay the most "attention" to. We are specifically using BERT for binary sequence classification, which refers to predicting a label (i.e., classification) for a sequence of words. In our case, during fine-tuning (i.e., training the model further on our own dataset) we will feed the

model pairs of sentences (where one sentence is the UPC description of a food item, and the other is the EC description of another food item), and the model will perform binary classification, predicting whether the sentences are a match (i.e., 1) or not (i.e., 0). We start with the 2015-2016 ground truth PPC data for positive examples, and five randomly sampled negative examples per positive example.

3 Related Work

Within the past few years, several papers have come out that learn *contextual* representations of sentences, where the entire sentence is used to generate embeddings. ELMo [12] uses a linear combination of vectors extracted from intermediate layer representations of a bidirectional LSTM trained on a large text corpus as a language model; in this feature-based approach, the ELMo vector of the full input sentence is concatenated with the standard context-independent token representations and passed through a task-dependent model for final prediction. This showed performance improvement over state-of-the-art on six NLP tasks, including question answering, textual entailment, and sentiment analysis. On the other hand, the OpenAI GPT [13] is a fine-tuning approach, where they first pre-train a multi-layer Transformer [14] as a language model on a large text corpus, and then conduct supervised fine-tuning on the specific task of interest, with a linear softmax layer on top of the pre-trained Transformer. Google's BERT [1] is a fine-tuning approach similar to GPT, but with the key difference that instead of combining separately trained forward and backward Transformers, they instead use a *masked* language model for pre-training, where they randomly masked out input tokens and predicted only those tokens. They demonstrated state-of-the-art performance on 11 NLP tasks, including the CoNLL 2003 named entity recognition task, which is similar to our semantic tagging task. Finally, many models have recently been developed that improve upon BERT, including RoBERTa (which improves BERT's pre-training by using bigger batches and more data) [11], XLNet (which uses Transformer-XL and avoids BERT's pretrain-finetune discrepancy through learning a truly bidirectional context via permutations over the factorization order) [15], and ALBERT (a lightweight BERT) [10].

In our prior work on language understanding for nutrition [8, 6, 2, 3, 9, 4, 5], we used a similar binary classification approach for learning embeddings, which were then used at test time to map from user-described meals to USDA food database matches, but with *convolutional* neural networks (CNNs) instead of BERT. The model was composed of a shared 64-dimension embedding layer, followed by one convolution layer above the embedded meal description, and max-pooling over the embedded USDA food name. The text was tokenized using spaCy. The meal CNN computed a 1D convolution of 64 filters spanning a window of three tokens with a rectified linear unit (ReLU) activation. During training, both the USDA input's max-pooling and the CNN's convolution over the meal description were followed by dropout of probability 0.1,[1] and batch normalization to maintain a mean

[1]Performance was better with 0.1 dropout than 0.2 or no dropout.

near zero and a standard deviation close to one. A dot product was performed between the max-pooled 64-dimension USDA vector and each 64-dimension CNN output of the meal description. Mean-pooling[2] across these dot products yielded a single scalar value, followed by a sigmoid layer for final prediction.[3] Further work demonstrated that BERT outperformed CNNs on several language understanding tasks, including nutrition [7].

4 Training Methods

Since we used a neural network model, the only features passed into our model were the tokenized words themselves of the EC and UPC descriptions—we did not conduct any manual feature engineering. The model was trained on a 90/10 split into 90% training and 90% validation data, where the validation data was used as a test set to fine-tune the model's parameters. We started with a randomly sampled set of 16,000 pairs, batch size of 16, AdamW as the optimizer, a linear schedule with warmup, and 1 epoch. We then added the next randomly sampled set of 16,000 pairs to get a model trained on 32,000 datapoints. Finally, we reached a total of 48,000 data samples used for training. Each pair of sequences was tokenized with the pre-trained BERT tokenizer, with the special CLS and SEP tokens, and was padded to the maximum length input sequence of 240 tokens.

5 Model Development Approach

We faced many challenges due to the secure nature of the ADRF environment. Since our approach relies on BERT, we were blocked by errors due to the local BERT installation. Typically, BERT is downloaded from the web as the program runs. However, for this challenge, BERT must be installed locally for security reasons. To fix the errors, the BERT models needed to be installed with git lfs clone instead of git.

Second, we were unable to retrieve the test data from the database due to sqlalchemy errors. We found a workaround by using DBeaver directly to save database tables as Excel spreadsheets, rather than accessing the database tables through Python.

Finally, we need a GPU in order to efficiently train our BERT models. However, we initially only had a CPU, so there was a delay due to setting up the GPU configuration. Once the GPU image was set up, there was still a CUDA error when running the BERT model during training. We determined that the model was too big to fit into GPU memory, so we found a workaround using gradient checkpointing (tradining off computation speed for memory) with the transformers library's Trainer and TrainingArguments. Unfortunately, the version of transformers we were using did not have these tools, and the library was not

[2]The inverse (mean-pooling before dot product) hurt performance.

[3]Note that our approach would work for newly added database entries, since we could feed the new database food's name into the pre-trained CNN to generate a learned embedding. This is the strength of using a binary prediction task, rather than a softmax output, so we do not have to re-train the network every time the database adds a new entry.

updated until less than a week before the deadline, so we still had to train the model on the CPU.

To deal with the inability to run jobs in the background, our process was checkpointing our models every five batches, and saving the model predictions during evaluation to a csv file every five batches as well.

6 Model Performance

After training, the 48K model was used at test time via ranking all possible 2017-18 EC descriptions given an unseen UPC description. The rankings were obtained through the model's output value—the higher the output (or confidence), the more highly we ranked that EC description. To speed up the ranking process, we used blocking (i.e., only ranking a subset of all possible matches), specifically with exact word matches (using only the first six words in the UPC description, which appeared to be the most important), and fed all possible matches through the model in one batch per UPC description. Since we still did not have sufficient time to complete evaluation on the full set of test UPC descriptions, we implemented an expedited evaluation that only considered the first 10 matching EC descriptions in the BERT ranking process (which we call BERT-FAST). We also report results for the slower evaluation method that considers all EC descriptions that match at least one of the first six words in a given UPC description, but note that these results are based on just a small subset of the total test set. See Table 1 below for our results, where the @5 indicates how often the correct match was ranked among the top-5. See Table 2 for an estimate of how long it takes to train and test the model on a CPU.

Table 1: S@5 and NCDG@5 for BERT, both for fast evaluation over the whole test set, anad slower evaluation on a smaller subset (711 UPCs out of 37,693 total).

Model	**Success@5**	**NDCG@5**
BERT-FAST	0.057	0.047
BERT-SLOW	0.537	0.412

Table 2: An estimate of the time required to train and test the model.

	Time
Training (on 48K samples)	16 hours
Testing (BERT-FAST)	52 hours
Testing (BERT-SLOW)	63 days

7 Suggestions for Future Challenges

We recommend that future challenges provide every team with both a CPU and a GPU in their workspace, to avoid transitioning from one to the other midway through the challenge. In addition, if possible, it would be very helpful to provide a mechanism for running jobs in the background. Finally, it may be useful for teams to submit snippets of code along with library package names, in order for the installations to be tested properly beforehand.

References Cited

[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Mandy Korpusik, Zachary Collins, and Jim Glass. Semantic mapping of natural language input to database entries via convolutional neural networks. *Proceedings of IEEE Conference on Acoustics, Speech and Signal Processing*, pages 5685–5689, 2017.

[3] Mandy Korpusik and Jim Glass. Spoken language understanding for a nutrition dialogue system. *IEEE Transactions on Audio, Speech, and Language Processing*, 25:1450–1461, 2017.

[4] Mandy Korpusik and Jim Glass. Convolutional neural networks and multitask strategies for semantic mapping of natural language input to a structured database. In *Proceedings of IEEE Conference on Acoustics, Speech and Signal Processing*, pages 6174–6178. IEEE, 2018.

[5] Mandy Korpusik and Jim Glass. Deep learning for database mapping and asking clarification questions in dialogue systems. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2019.

[6] Mandy Korpusik, Calvin Huang, Michael Price, and Jim Glass. Distributional semantics for understanding spoken meal descriptions. *Proceedings of 2016 IEEE Conference on Acoustics, Speech and Signal Processing*, pages 6070–6074, 2016.

[7] Mandy Korpusik, Zoe Liu, and James Glass. A comparison of deep learning methods for language understanding. *Proc. Interspeech 2019*, pages 849–853, 2019.

[8] Mandy Korpusik, Nicole Schmidt, Jennifer Drexler, Scott Cyphers, and Jim Glass. Data collection and language understanding of food descriptions. *Proceedings of 2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 560–565, 2014.

[9] Mandy Korpusik, Zachary Collins, and Jim Glass. Character-based embedding models and reranking strategies for understanding natural language meal descriptions. *Proceedings of Interspeech*, pages 3320–3324, 2017.

[10] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

[11] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[12] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[13] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*, 2018.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, 2017.

[15] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764, 2019.