Geerthan Srikantharajah
500839837

Bi-GRU for Aspect-Based Sentiment Analysis with Auxiliary Data

Introduction

Natural language processing is a very complicated, yet crucial field. While it encompasses text generation, understanding text is the first step to many NLP tasks. For example, reading comprehension questions involve an understanding of the text to find the answer. This project focuses on sentiment analysis, where an understanding of the text is required to determine whether a text possesses a positive or negative sentiment. Instead of limiting sentiment to words or sentences, we usually associate sentiment with subjects within a sentence. For example, in the sentence “my dinner was great”, the subject would be “my dinner” and the associated sentiment would be positive. This is the basis of sentiment analysis.

Aspect-based sentiment analysis (ABSA) takes the dissection of sentiment one step further. The above sentiment would be considered a general sentiment, because the sentence describes the subject positively as a whole. No specific aspect of the subject is mentioned with any sentiment. In the sentence “my dinner tasted great, but it was too expensive”, two different aspects of the subject are mentioned, with different sentiments. Taste and price are the two mentioned aspects of the dinner, where taste has a positive sentiment, but cost has a negative sentiment. With regular sentiment analysis, the overall result would not be able to represent the multiple sentiments we see through aspect-based sentiment analysis. There are many tasks that can be automated with this. For example, you could automate the processing of product reviews and determine which aspects of a product people like and dislike.

There are a few ways of approaching sentiment analysis. Some groups have made lexicons which map words to sentiments. An example would be AFINN [1], by F. Nielsen. A very rudimentary approach would be to average all these mappings within a sentence. One early solution by Wagner *et al.* [2] combined the use of a sentiment lexicon with an SVM, performing feature engineering in order to generate features for use with the SVM. A basic method is used to handle negation, where if a negator word (such as “not”) is found in close proximity to a word with a sentiment, its sentiment is inverted. Newer methods involve the use of neural networks or language models (such as BERT [3]), with improvements in accuracy.

Word embeddings are powerful tools when it comes to NLP, and sentiment analysis is no exception. My approach to this problem is to use word embedding information with a bi-directional GRU. The use of gated recurrent units provides a similar benefit to LSTMs (with long and short term memory), where sequential data (such as sentences) can be analyzed, with the relevant information being stored at each processing step. The bi-directional GRU also possesses an attention layer to prioritize relevant words in each sentence. The resulting features are fused with sentiment lexicon data before each subject-aspect sentiment is classified.

Method

The proposed network acts on sentences. The training data for each sentence contains a sentiment value for each subject-aspect pair, where the subjects and aspects are constants across all data. The sentiment value for each subject-aspect pair is one of [None, Positive, Negative], so the task is a multi-class classification problem. A separate network is trained for each subject-aspect pair, breaking up the task into subtasks where each network only needs to predict one sentiment. This massively simplifies the required task. Identifying subjects and aspects are not within the scope of the task, as lists of subjects and aspects are used. This is also for the sake of simplicity.

First, word embeddings are generated for each word within the sentence. Simultaneously, sentiment values from the AFINN sentiment lexicon [1] are retrieved for each word. Words without a sentiment are given a zero value. Word embeddings provide unique representations of words, which can be used to analyze their meanings as well as relations between words. Sentiment values, on the other hand, provide direct representations of a word's sentiment. The AFINN lexicon provides values from -5 (very negative) to 5 (very positive). The addition of sentiment values provides useful auxiliary information to the model.

The word embeddings are sent into the bidirectional GRU (with attention) for each subtask, to be independently trained and tested. The resulting output from different word locations is acquired (depending on which model is tested), as well as the attention weights from those locations. These attention weights are used to weigh the previously generated sentiment values, which are then fused with the GRU output. A fully connected layer and softmax classifier are used in order to predict the final sentiment.

A GRU allows for the processing of sequential data, with the benefit of being able to forget irrelevant information. It also uses less memory than an LSTM. The attention weighting provides insight (using the GRU) into the significance of each word.

Several different methods of sampling from the GRU were tested. The first model (AbsaGRU-Original) uses the output from the GRU at the location of the subject word. The second model (AbsaGRU-Last) uses the output from the GRU at the location of the final word in the sentence. Some sentences possessed two mentions of a subject, but models that fused the GRU values from both locations did not perform well.

Implementation

main.py

The main python file trains a model for one subject/aspect pair, and generates output in a text file. The implementation consists of three sections. The first section is data and model setup. The following actions are performed within the first section:

- Parameter initialization (ex. Subject name, aspect name, batch size, embedding dimension)
- Tokenizer setup (Basic tokenizer from torchtext, generates words from a sentence)
- Sentiment lexicon loading (AFINN-111)
    - Dictionary returns value between [-5, 5] when a word is provided
- Data loading, from ABSAData.py

- PyTorch DataLoader setup
    - Note: the dataset mainly contained subject-aspect pairs with the sentiment of None
    - A Weighted Random Sampler is used for training to provide a roughly equal amount of samples per sentiment class, within each epoch
    - Performed slightly better than random sampling
- Model, loss, and optimizer initialization
    - Cross entropy loss is used
    - Adam optimizer used (with a learning rate of 0.001)

The second section consists of model training. The train set is used to train the model, and the dev set (or validation set) is used to evaluate the loss and accuracy of the model at each epoch. The best model is picked based on its loss with regards to the dev set, to prevent overfitting.

The third section loads the best model and test dataset, and then generates sentiment predictions for each sentence. These predictions are written to a text file.

ABSAData.py

This file processes the Sentihood [4] dataset. The dataset is split up into folders for each subject/aspect pair, and each folder consists of a tsv file for training (train), validation (dev), and testing (test). I limited the maximum sentence length for the GRU to 128 words. The get_train and get_other functions read the tsv files, and then process them by calling process_train_tsv or process_tsv respectively.

The process_tsv functions share a lot of the same code. First, sentences have the corresponding subject tokens (“location - 1" and “location - 2") replaced with one word tokens, “loc1” and “loc2”. At this next stage, the process_train_tsv function differs. GloVe word embedding vectors are loaded from torchtext. The vocab variable is used to map words to unique IDs, where the IDs are then mapped to word embeddings with the pretrain_embed matrix. The pretrain_embed variable is used to provide word embeddings to the nn.Embedding module within PyTorch later on.

After this point, the two methods match almost entirely. Each sentence is converted into the required data for the model. Within each sentence, vocab[sentence] represents the word embedding for each word, and sen_id contains the list of word embeddings for the current sentence. Sentiments are acquired from the afinn_dict for each word and stored in sen_sent. Subject locations are stored in subj_loc. These are all padded to their respective maximum lengths. After this, sentiment labels are converted to numbers from [0,2] with labelMap. Finally, the lists are converted to Tensors and merged into a TensorDataset for use with PyTorch’s DataLoader.

One more difference at the end of process_train_tsv is that a train_weights list is generated. This weight list is required for the WeightedRandomSampler that is being used on the training dataset. Essentially, each class gets a weight based on the inverse of how frequent it is. Sentences with a sentiment of “None” get a very low weight because they appear very frequently, and the other sentences have high weights.

ABSAModel.py

This file contains the AbsaGRU model. The AbsaGRU model represents the full network used for classification. It takes four tensors as input: one for word IDs, one for sentence sentiment labels (for training), one for subject locations, and one for word sentiment labels. The embedding layer converts the word IDs into word embeddings. A dropout layer is used to encourage a generalized model. Then, PyTorch's GRU layer is used in bidirectional mode. After this, a multihead attention layer is used for single head attention (the number of heads is a parameter). The output as well as the attention weights are generated. Both of these are generated for each word location within the sentence, so the desired word location is then picked with the subs matrix. The sentiments are then weighted by the attention weight matrix. The GRU output and weighted sentiment data are sent to a dropout layer, and then merged into one large feature vector. This vector is normalized and sent to a fully connected layer, with an output size of 3 (for the three possible sentiment classes).

evaluation.py

This file is a trimmed down version of the evaluation file used in Sun *et al.* [3]'s testing. It measures five different criteria.

testSentihood.bat

This batch file runs main.py (training and testing) for all possible subject/aspect pairs within Sentihood, and then evaluates the result with evaluation.py. The results are shown in the console window.

Evaluation

There were five different evaluation metrics used for this problem. For aspect evaluation, strict accuracy (acc), F1 score (F1), and AUC (area under the ROC curve) were used. For sentiment evaluation, strict accuracy and AUC were used. These are the same measurements used by Sun *et al.* [3]. In table 1, the models shown are the AbsaGRU-Original model, which uses the feature vector from the subject's first location, and the AbsaGRU-Last model, which uses the feature vector from the end of the sentence. The BERT results are shown with the BERT-pair-NLI-M mode, where the size of the BERT model is varied. The AbsaGRU results are averaged over five models. The best results are in bold and italics, and the second-to-best results are in bold. In table 2, the BERT-tiny modes are compared.

Model	Aspect			Sentiment	
	Accuracy	F1 score	AUC	Accuracy	AUC
BERT-tiny	59.6	48.4	87.7	80.0	77.2
BERT-medium	***72.0***	***80.9***	***95.5***	***87.4***	***91.9***
AbsaGRU-Original	**61.3**	**69.9**	**90.7**	**83.6**	85.8
AbsaGRU-Last	60.1	68.7	**90.7**	83.3	**87.2**

Table 1: BERT-pair-NLI-M compared to AbsaGRU

Model	Aspect			Sentiment	
	Accuracy	F1 score	AUC	Accuracy	AUC
NLI_M	59.6	48.4	87.7	80.0	77.2
NLI_B	53.3	29.3	83.7	76.0	70.9
QA_M	59.7	46.8	88.7	80.9	77.7
QA_B	48.4	1.3	57.7	67.7	55.1
AbsaGRU-Original	**61.3**	**69.9**	**90.7**	**83.6**	85.8
AbsaGRU-Last	60.1	68.7	**90.7**	83.3	**87.2**

Table 2: BERT-tiny compared to AbsaGRU

Discussion

Within table 1, we see a clear advantage for BERT-medium on this task. However, there is an important difference between these models. BERT-medium took 2 hours and 20 minutes to train on an NVIDIA GeForce RTX 3060 (Laptop), while the GRU models took 6 minutes to train. BERT-tiny took 38 minutes to train, which is significantly shorter than BERT-medium, but was outperformed by AbsaGRU in all metrics. A larger BERT model is needed to learn the data.

BERT-medium has several advantages compared to AbsaGRU in this study, leading to its success. The pretrained BERT model has an inherent advantage from the start, though this can be alleviated with enough training. However, it is difficult to train AbsaGRU enough on the Sentihood dataset to match the knowledge base that BERT has. AbsaGRU requires splitting the task into subtasks for each subject/aspect pair (8 total), meaning that the small Sentihood dataset gets divided even further. There are not enough samples for the subtasks to have sufficient training data. Overfitting on the training set happened constantly, and many steps had to be taken in order to prevent it, such as reducing the network parameters and adding dropout layers. On the other hand, the BERT model is trained on every subject/aspect pair simultaneously, and the problem setup is significantly different as BERT acts on sentence pairs with generated sentences. The excess of training data and the fundamentally different problem that the BERT model is tasked with leads to a significant advantage, on top of the architectural benefits of the BERT structure, including the use of subwords as well as additional data for each word embedding.

From the evaluation metrics, taking the feature vector from the subject location (AbsaGRU-Original) looks better than taking the feature vector from the end of the sentence. However, these metrics are still very close to each other. Some of the AbsaGRU-Last models outperformed some of the AbsaGRU-Original models as well. There does not appear to be any major difference between the two AbsaGRU methods.

Citations

[1] F. Å. Nielsen, “A new ANEW: Evaluation of a word list for sentiment analysis in microblogs.” arXiv, Mar. 15, 2011. doi: 10.48550/arXiv.1103.2903.

[2] J. Wagner et al., “DCU: Aspect-based Polarity Classification for SemEval Task 4,” in Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014), Dublin, Ireland, Aug. 2014, pp. 223–229. doi: 10.3115/v1/S14-2036.

[3] C. Sun, L. Huang, and X. Qiu, “Utilizing BERT for Aspect-Based Sentiment Analysis via Constructing Auxiliary Sentence,” in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, Minnesota, Jun. 2019, pp. 380–385. doi: 10.18653/v1/N19-1035.

[4] M. Saeidi, G. Bouchard, M. Liakata, and S. Riedel, “SentiHood: Targeted Aspect Based Sentiment Analysis Dataset for Urban Neighbourhoods,” in Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, Osaka, Japan, Dec. 2016, pp. 1546–1556. Accessed: Nov. 07, 2022. [Online]. Available: https://aclanthology.org/C16-1146