Twitter Airline Sentiment Analysis using ULFMiT

Chenyang Yu

March 2020

Abstract

We present the capstone project report for the Machine Learning Nanodegree from Udacity. We have borrowed the Universal Language Model Fine-tuning (ULFMiT) algorithm originally developed by Jeremy Howard from fast.ai and Sebastian Ruder, and adapted to analyze Twitter airline sentiments. ULFMiT is an effective transfer learning method that can apply existing pre-trained models to any related task in NLP. Our tweets data on airline sentiments are provided by Kaggle challenge.

Keywords: ULFMiT, Transfer Learning, NLP

1 Definition

1.1 Overview

Natural language processing (NLP) is the technology used to aid computers to understand the human's natural language. The objective of NLP is to read, decipher, understand, and even generate human languages in a manner that is valuable. It started in the 1950s, although works can be traced as early as the Turing Test. It is one of the most important fields in machine learning and AI. NLP is commonly seen in Interactive Voice Response (IVR), language translation, grammar checker, and etc.

Machine learning techniques were used to be considered for NLP tasks, while deep learning is traditionally used for image processing, and classification. However, in recent times, deep learning is used to classify topics, tag sequences, analyze sentiments, and generate natural language. Deep learning methods such as ULMFiT, OpenAI GPT, ELMo and Google AI's BERT have revolutionized the field of transfer learning in NLP. We are able to deploy pre-trained deep learning models that have the ability to transfer knowledge learned from large curated datasets to solve other NLP problems. Transfer learning has shown great improvements in semantic understanding of language.

ULFMiT utilizes inductive transfer learning to apply AMD-LSTM (Merity et al., 2017a), a regular LSTM model with fine-tuned hyperparameters. The base architecture of ULMFiT is ASGD Weight-Dropped LSTM (AWD-LSTM), which implements dropouts, BPTT, and multi-batch encoder. ULFMiT approach is semi-supervised learning since the AWD-LSTM model was pre-trained in an unsupervised manner, but then fine-tuned in a supervised manner. In the Twitter sentiment analysis, I implement the same approach.

1.2 Problem Statement

The project is aiming to analyze what feelings Twitter users expressed by tweeting in February 2015, through creating and tuning an ULMFiT model to correctly classify Twitter airline sentiment data. The result should produce 1 of 3 sentiments (positive, negative, and neutral). The tweets are quantifiable through tokenization to create a numeric representation of words. I will be using the ULFMiT language model provided from the fast.ai library and applying the pre-calculated weights. It will provide a word embedding scheme that aligns with the corpus of airline tweets. In order to capture the meaning in each word, the language model hyperparameters will be tuned and it will lead to a sentiment result. If the model is accurate, my result will match the pre-labeled sentiment.

1.3 Metrics

a. F1 Score

Evaluation will be using the F1 score (F1μ) for the three sentiment classes - Positive, Negative and Neutral on the submissions made with predicted class of each sample in the evaluation data set. The calculation of F1 score is the harmonic mean of precision and recall:

$$F_1 = \left(\frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} \right) = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

b. The error rate = 1 – Accuracy

This is the error rate used in ULMFiT paper. I am able to compare their validation error benchmark on the IMDb sentiment classification task.

1.4 Benchmark

Mentioned in the ULMFiT paper (Howard and Ruder, 2018), the base architecture of ULMFiT is ASGD Weight-Dropped LSTM (AWD-LSTM), which implements dropouts, BPTT, and multi-batch encoder. In addition, ULMFiT process includes LM pre-training, LM fine-tuning using novel techniques Discriminative fine-tuning and Slanted triangular learning rates, and finally classifier fine-tuning using Gradual unfreezing, BPTT, and bi-directional LM. The optimization techniques mentioned above were added to the benchmark model one by one. The final model implemented on IMDb dataset resulted in a 5% benchmark validation error rate.

2 Analysis

2.1 Data Exploration and Pre-processing

We used Twitter airline sentiment data that contains 14640 labeled tweets directed at six major US airline companies (Virgin America, United, Southwest, Delta, US Airways and American Airlines). Twitter dataset came in both .csv and database format since it is obtained from a Kaggle challenge, originally came from Crowdflower's Data for Everyone library. Tweets were scraped from February of 2015 and it includes labeling of positive, negative, and neutral sentiments. Figure 1 shows 5 tweets samples and their sentiments stored in Pandas DataFrame.

	airline_sentiment	text
7717	neutral	@JetBlue Re: Flight 8088 SXM>JFK what time ...
13854	negative	@AmericanAir why are you still selling tickets...
11719	negative	@USAirways I have now called 12 times in the l...
239	positive	@VirginAmerica sounds like fun !
8582	neutral	@JetBlue Statement on #Lufthansa Incentive Off...

Figure 1: Sample tweets

2.1.1 Data Exploration

Twitter sentiment contains more negative comments than neutral and positive tweets combined. The Class Imbalance could be the cause of data collection. United airlines, US Airway, and American airlines have the most tweets. Virgin America has the least tweets. The total number of positive and neutral tweets are less than 50% of the total tweets per airline. However, figure 2 shows that is not the case for 3 airlines, Delta, Southwest and Virgin airlines. One reason could be the limited number of tweets in the dataset while most tweets are negative.

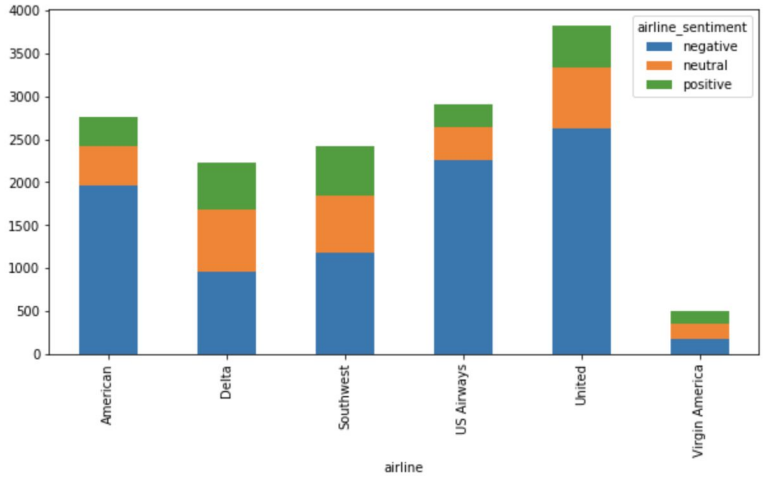

Figure 2: Number of Airline Tweets by Sentiments

2.1.1 Data Pre-processing

We removed non-ASCII chars from texts since the ULFMiT model is trained and tested on English texts. Additionally, dataset was split into training set (70%) and testing set (30%). 10248 tweets are in the training set, and 4392 tweets belong to the testing set.

2.2 ULFMiT

ULFMiT utilizes inductive transfer learning to apply AMD-LSTM to develop a language model called Wikitext-103 and was trained of 28,595 preprocessed Wikipedia articles. Specifically, it follows 3 key steps: general-domain LM pre-training, target task LM fine-tuning, and lastly, target task classifier fine-tuning. We are implementing LM fine-tuning and Classifier fine-tuning as seen in Figure 3 since the first stage LM pre-training is computational expensive that we will implement transfer learning.

For LM fine-tuning, ULMFiT process includes novel techniques Discriminative fine-tuning and Slanted triangular learning rates. During classifier fine-tuning, it implements Gradual unfreezing, BPTT, and bi-directional LM.

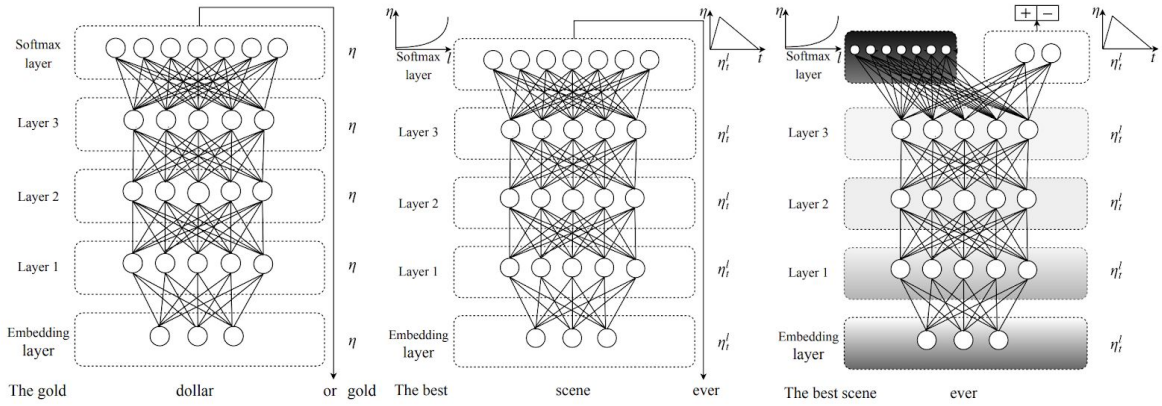

Figure 3: ULFMiT architecture (ULMFiT paper by Jeremy Howard and Sebastian Ruder)

2.2.1 LM fine-tuning

In the ULFMiT package, TextLMDataBunch returns a "Databunch" type of object and helps assembling the raw data to be suitable for NLP. It is used to transform dataframe, csv, tokens, and other data forms into the right form to be used for deep learning. Additionally, The Databunch code includes text manipulation code from the ground up and tokenization. We define a databunch object (data_lm) using training data. **Figure 4** shows 5 sample tweets after transformed into "Databunch" type of object. A total of 5008 vocab were obtained. In NLP tasks, batch size is the first dimension and sequence length is the second because we don't use the sequence length as the first dimension.

idx	text
0	i 'm willing to wait on hold , but that 's not an option . xxbos @virginamerica sounds like fun ! xxbos @jetblue xxmaj statement on # xxmaj lufthansa xxmaj incentive xxmaj offer - xxmaj xxunk http : / / t.co / xxunk xxbos @americanair xxmaj thx ! i hope so . xxup iah to xxup dfw to xxup okc has turned out to be a xxup long trip today
1	ground staff , it has n't even been taken off the plane yet ! xxbos @usairways i am following you now xxbos @jetblue after my second call to customer service and the xxunk person i talked to , an amazing rep fixed it in about 5 mins ! :) # xxunk xxbos @southwestair so for an extra luggage , it cost $ 75 xxrep 4 ? xxbos @usairways xxmaj will
2	10 flights have been delayed . xxmaj huge mistake on my part . xxmaj do n't fly united xxbos @jetblue anything for you . # flyfi http : / / t.co / xxunk xxbos @united even so , change could not be made xxunk instructed to call an xxunk maybe the xxunk should allow agents 2 link passengers ! xxbos @americanair great , thanks ! xxbos @jetblue tells me $
3	failure in 2015 # unitedairlines xxbos @jetblue 's xxup ceo xxmaj battles to xxmaj appease xxmaj passengers and xxmaj wall xxmaj street - http : / / t.co / e5naxbue4s http : / / t.co / xxunk xxbos @united what 's the best way to get your tickets ? xxmaj print off at home or go to check in desk ? xxbos @united i would take the $ 500 voucher
4	yesterday . i wonder how quickly flight attendants are notified . xxbos @americanair i dmed xxbos @jetblue what is the deal with flt xxunk today ? xxmaj departure keeps changing . xxmaj when is it going why is it so xxmaj late xxmaj flight ? xxbos xxmaj yes but i will nvr fly w / @usairways i missed my con flight bc of a xxunk on xxup xxunk xxmaj could

Figure 4: Databunch tweets

We define an AWD_LSTM learner object (learn_AWD) that uses the tokenized language model data, using function language_model_learner. We chose 0.5 (drop_mult) as the multiplier applied to the dropout in the language learner. **Figure 5** demonstrates the architecture of AWD_LSTM.

```
[AWD_LSTM(
   (encoder): Embedding(5008, 400, padding_idx=1)
   (encoder_dp): EmbeddingDropout(
     (emb): Embedding(5008, 400, padding_idx=1)
   )
   (rnns): ModuleList(
     (0): WeightDropout(
       (module): LSTM(400, 1152, batch_first=True)
     )
     (1): WeightDropout(
       (module): LSTM(1152, 1152, batch_first=True)
     )
     (2): WeightDropout(
       (module): LSTM(1152, 400, batch_first=True)
     )
   )
   (input_dp): RNNDropout()
   (hidden_dps): ModuleList(
     (0): RNNDropout()
     (1): RNNDropout()
     (2): RNNDropout()
   )
 ), LinearDecoder(
   (decoder): Linear(in_features=400, out_features=5008, bias=True)
   (output_dp): RNNDropout()
 )]
```

Figure 5: AWD_LSTM Architecture

Fastai package provides a convenient way to find the optimal learning rate. For our optimization function, the learning rate needs to be at least an order of magnitude below the point at which the loss starts to diverge. Discriminative fine-tuning per the ULMFiT paper, allows different learning rates to be applied to each layer. **Figure 6** (left) shows that the learning rate finder trains the model over the duration of 1 epoch at a very low learning rate to begin with, and gradually increases learning rate over time.

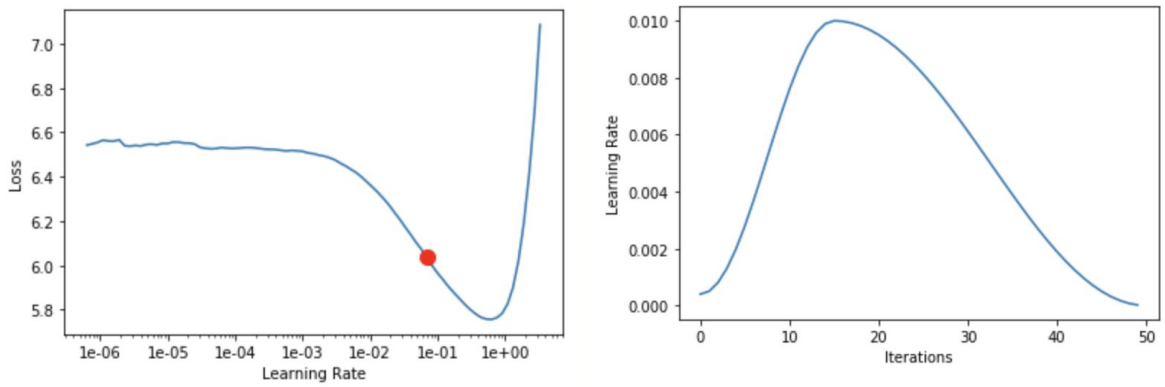

Figure 6: Learning Rate Finder

Figure 6 (right) demonstrates the behavior of slanted triangular learning rates per the ULMFiT paper. The learning rate was linearly increased then linearly decayed, in order to find the optimal. It helps the model to quickly adapt parameters to task-specific features. It shortens the time for the model to converge to a suitable region in the beginning of training and then refines its parameters. Figure 6.1 demonstrates the point of the angle when we switch from increasing to decreasing the LR, according to Smith's paper on Cyclical Learning Rates for Training Neural Networks.

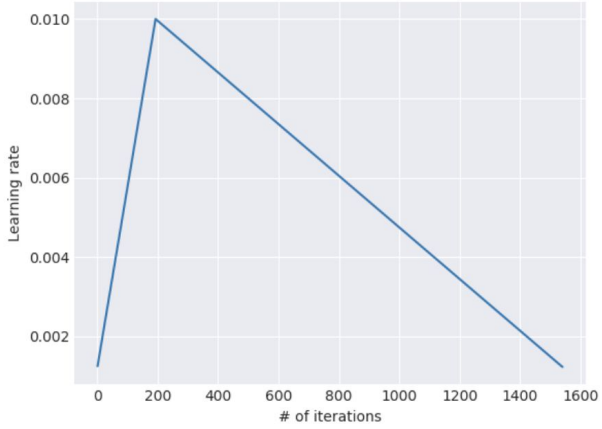

Figure 6.1: Slanted triangular learning rates

We modified the learning rates and ran the language model fine-tuning step until the validation loss drops to a low value. Next, we fit the model for a few cycles by running 1 epoch. In **figure 7**, we unfreeze all the layers and then run more epochs for a reasonable time to fine tune until we obtain a low enough validation loss. Since the vocab dictionary is relatively small, we can see validation loss decreasing to a local minimum quickly (**Figure 7**).

```
[ ]  learn_AWD.unfreeze()
     learn_AWD.fit_one_cycle(10, 1e-3, moms=(0.8,0.7))
```

epoch	train_loss	valid_loss	accuracy	time
0	4.022155	3.885998	0.264211	00:10
1	3.899882	3.769915	0.279390	00:10
2	3.751401	3.709435	0.282850	00:10
3	3.559939	3.672695	0.291592	00:10
4	3.335809	3.688039	0.290960	00:11
5	3.101495	3.717893	0.289844	00:11
6	2.896253	3.780837	0.286421	00:11
7	2.701619	3.838577	0.284747	00:11
8	2.554755	3.872753	0.284747	00:11
9	2.479051	3.883733	0.284003	00:11

Figure 7: Unfreeze all the layers and then run more epochs

2.2.2 Classifier fine-tuning

For classification, we load the data for the forward classifier model using the same language model vocabulary, using TextList data loader. Hyperparameters databunch batch is set to 128, and split (split_by_rand_pct) 10% to validation set and 90% to training set. Similar to learn_AWD learner object, we define a classifier learner (learn_clas) using text_classifier_learner funcion from fast.ai library.

Optimal learning rate was found using LR finder. For training the classifier, we apply "gradual unfreezing", mentioned in ULFMiT paper. The paper proposed to gradually unfreeze the model starting from the last layer and fine-tune all unfrozen layers in one epoch. The latter a layer is, the least memory it has compared to other layers. The "gradual unfreezing" technique then freezes the next lower layer and repeats the fine-tuning process until convergence at last iteration. During gradual unfreezing, we apply discriminative fine-tuning on learning rates. ULFMiT (Howard et al., 2018) recommends to choose the learning rate ηL of the last layer by fine-tuning only the last layer and using $\eta L-1 = \eta L/2.6$ as the learning rate for lower layers. Slice function in **figure 8** applies different learning rates for different learning groups. As in figure 8, validation loss gradually decreases while accuracy increases.

```
lr=2e-2
learn_clas.fit_one_cycle(1, lr, moms=(0.8,0.7), wd=0.1)
learn_clas.save('first')
```

epoch	train_loss	valid_loss	accuracy	time
0	0.695790	0.588845	0.764648	00:06

```
lr/=2
learn_clas.freeze_to(-2)
learn_clas.fit_one_cycle(1, slice(lr/(2.6**4),lr), moms=(0.8,0.7), wd=0.1)
learn_clas.save('second')
```

epoch	train_loss	valid_loss	accuracy	time
0	0.643554	0.547854	0.787109	00:06

```
lr/=2
learn_clas.freeze_to(-3)
learn_clas.fit_one_cycle(1, slice(lr/(2.6**4),lr), moms=(0.8,0.7), wd=0.1)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.575164	0.511689	0.794922	00:06

```
lr/=5
learn_clas.unfreeze()
learn_clas.fit_one_cycle(4, slice(lr/(2.6**4),lr), moms=(0.8,0.7), wd=0.1)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.504700	0.503367	0.809570	00:06
1	0.477284	0.485592	0.807617	00:06
2	0.469016	0.485677	0.813477	00:06
3	0.441277	0.482229	0.815430	00:06

Figure 8: Gradually unfreezing layers while decreasing learning rates

2.2.3 Bidirectional Language Model

In order to apply a bidirectional language model, we'll reverse the words in tweets from the training set. We'll follow the exact same steps we took for language model (data_lm) and language model learner (learn_AWD). We load data from the pre-saved databunch object (data_lm.pkl) where we save the training data, reverse it(Backwards=True), and save it into 'data_lm_back.pkl' for later access. Similar to the forward language model, we apply the language model, the learning rate finder and train 1 epoch for the backward data. We train a backward learner (learn_clas_back), and apply the gradual unfreezing and the discriminative fine-tuning.

2.2.3 Ensemble Model

For both forward and backward language models, we cross-tabulate the predictions for each class and calculate F1 score and accuracy in figure 9. Figure 10 shows that the forward model has a high accuracy.

Ensemble method is to combine the predictions from both models and calculate the average as the final result. Figure 10 also shows that the ensemble method increases the accuracy by 0.005.

```
preds_fwd, targets_fwd = learn_clas.get_preds()
predictions_fwd = np.argmax(preds_fwd, axis=1)
pd.crosstab(predictions_fwd, targets_fwd)
```

col_0	0	1	2
row_0			
0	583	66	29
1	35	129	16
2	17	26	123

```
preds_back,targets_back=learn_clas_back.get_preds()
predictions_back=np.argmax(preds_back,axis=1)
pd.crosstab(predictions_back,targets_back)
```

col_0	0	1	2
row_0			
0	582	71	29
1	40	125	19
2	13	25	120

Figure 9: Confusion matrix for the predictions

```
# get combined(mean) predictions
ensemble_preds = (preds_fwd + preds_back)/2
# get combined(mean) accuracy on validation set
print('Ensemble classifier results (validation set): \nValid
print('FWD classifier results (validation set): \nValidation
print('BACK classifier results (validation set): \nValidatio
```

```
Ensemble classifier results (validation set):
Validation accuracy: 0.8203, Validation error rate: 0.1797
FWD classifier results (validation set):
Validation accuracy: 0.8154, Validation error rate: 0.1846
BACK classifier results (validation set):
Validation accuracy: 0.8076, Validation error rate: 0.1924
```

Figure 10: Gradually unfreezing layers while decreasing learning rates

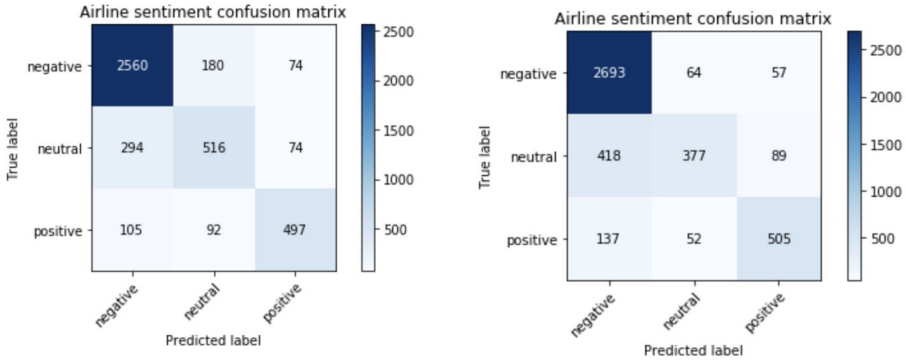

Figure 11: Confusion matrix for both models

2.3 Results

We tested both the forward and backward model on a testing set. **Figure 11** shows the plotted confusion matrices for both models, in order to analyze the misclassifications under each sentiment. It shows both

the test set's true labels and our model's predictions. Backward model (**Figure 11** right) has more accurate predictions on negative tweets. The forward model (**Figure 11** left) has more accurate predictions on neutral tweets. Both have F1 score 0.818, while the forward model has an error of 0.18 and the backward model has an error of 0.19. The increase of accuracy and F1 score from ensemble model is minimal in this case.

2.4 Comparison to benchmark model

Compared to the benchmark model the ULFMiT paper tested on IMDB reviews dataset with 5% validation error, our ensemble model has 18% validation error. Our model implemented similar techniques to the benchmark model. However, both models utilized transfer learning from the model that were pre-trained on wikipedia data with long paragraphs. It makes sense that it performs better on longer paragraphs like IMDB reviews than tweets limited to 140 words. The structure of Wikipedia paragraphs is different from tweets which are more similar to conversations. The use of concatenated pooling in the model also reduces the error when used on large sequences.

3 Conclusion

This project showed a training and classification pipeline for fastai's framework ULMFiT for evaluating Tweets sentiment. With some data cleaning and hyperparameter tuning, the transfer learning approach using ULMFiT can provide good classification accuracy and F-scores. ULMFiT is flexible and easy to perform classifier training and retraining with a small dataset. However, the language model fine-tuning is expensive and less effective since the language model was pre-trained on text data different from tweets or conversations.

4 Improvements

Improvements through the following modifications could be implemented in the future.

A. Vocabulary Augmentation:
   Although we implemented transfer learning from pre-trained deep learning model that was pretrained on the Wikipedia dataset, we could augment the dataset with extra Vocabulary. The vocabulary in the training set do not necessarily match the vocabs in our testing set. Specifically, Kaggle Twitter Sentiment140 dataset would be an ideal augment Vocabulary dataset to train our model on. It will allow our model to better understand the structure and specifications of tweets.
B. Data Collection:
   It would be more accurate to train the model on a larger generalized dataset that was collected during a span of time.
C. The Use of Concatenated Pooling:
   ULFMiT uses concatenated pooling for large sequences of text. For example, IMDB reviews dataset. However, limited to 140 characters, tweets do not have large sequences. If the use of concatenated pooling can be modified and improved, model performance will increase.

References

https://arxiv.org/pdf/1801.06146.pdf

https://arxiv.org/abs/1506.01186

https://towardsdatascience.com/transfer-learning-in-nlp-for-tweet-stance-classification-8ab014da8dde

https://docs.fast.ai/train.html

https://github.com/fastai/fastai/tree/050080be574cb1260462bbd03e9600e43e7a54b1