CS5740: Project X
Github Repository Link (final-rashmi59)

Rashmi Sinha
rs2584

1 Introduction (4pt)

We are required to develop a name-entity recognizer for Twitter. Given a tweet, we need to identify sub-spans of words representing named entities with high accuracy and high speed.The data is noisy, simulating real world with emojis and slangs commonly used in conversational english.

Transformer architecture based on multihead attention is the SOTA of many NLP tasks. I used pretrained transformer models trained on English tweets - BERTweet[3] to finetune on the data provided.

I tried different model architectures, training epochs, learning rates and inference methods. The best F1 score on dev set was 0.713 and test set was 0.674.

2 Task (3pt)

There are T tweets. Each tweet is tokenized into n tokens delimited by newline. $T = t_1, t_2..t_n$. t_i is associated with single NER label l_i from B, I, O, like $t_1 - B, t_2 - I$. B marks beginning of the token and I marks continuation. I cannot exist without B.

3 Data (4pt)

Only data provided was used. As I used a pretrained BerTweet model which is already trained on 850M cased tweets, extra data didn't seem needed. I did not apply any processing on the tokens and relied on the BertweetTokenizer for tokenization. Refer to table 1 for all statistics related to data. The maximum number of subtokens present for a tweet in the train dataset is 62. For batching, I padded any smaller example to length 62. I do not apply this for dev and test as predictions were done individually.

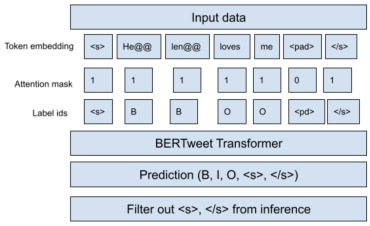
Dataset	no. of tweets	avg tweet length	no.of tokens	no.of tokens aft tokenization	avg tweet length aft tokenization
Train	2395	19.4	46469	61310	25.6
Dev	960	13.9	13360	21484	22.4
Test	2377	14.03	33354	52987	22.3

Table 1: Data Overview

4 Model (18pt)

I used the BerTweet model which uses the same architecture as the BERT-base[1], and is trained using the RoBERTa pre-training procedure [2]. The BERT base architecture consists of 12 layers (transformer blocks), 12 attention heads and 110 million parameters. First, I converted the tokens into multiple subtokens using the Bertweet-Tokenizer with default parameters. I consciously chose not use the normalization parameter in the tokenizer because 99% of the time, the emojis just acted as noise and were allocated 'O' tag. Without normalization, they are tokenized to ¡unk¿ token which is mapped to 'O' anyways. Adding normalization makes it difficult to map back the tag to the original format resulting in extra processing and hit on speed. As fast speed was a criteria and I was getting optimal results, I chose to ignore it. When a token is broken down into subtokens, the label ids of that token is replicated across subtokens. Eg: Bambi with tag 'I' is broken into subtokens Bam@@ and bi@@, each with label 'I'. As batching is used for training, all the input tensors in a single batch need to be of same length. All the tokens less than the maximum length are padded. Attention masks are used to enable the model to ignore the padded elements in the sequence. To finetune the BerTweet model, AdamW optimizer is used along with weight decay as regularization for weight matrices. AdamW is used instead of SGD as it works well with attention models[4]. Using weight decay is important given the nature of the problem as we do not want the model to overfit due to the multi-sense na-

Figure 1: Model architecture



ture of words depending on context, (eg: 'scrolls' can have both tag 'O' and 'B' as in dev). Linear scheduler is used to reduce the learning rate through the epochs for convergence. Lastly, the norm of the gradient is clipped to prevent the exploding gradients problem. Finally, the encoded subtoken ids, label ids and the attention masks are inputted into the model and fine-tune it on the entire training data. (Refer Figure 1). For the inference, each tweet is predicted individually by concatenating tokens into a single sentence delimited by space. The sentence, encoded using the BertweetTokenizer, is fed into the model and the label ids are decided based on the maximum probability. Finally the subtoken is mapped back to the tokens, by having 2 parallel pointers to the original sentence and the subtokenized sentence. Any subtokenized word is appended with '@@' which helps identify and combine tokens together. As there can have multiple tags, the final tag of the token in the order of priority of 'B', 'I' and lastly 'O'. BERTweet relies on byte-pair encoding(BPE). BPE is a frequency-based character concatenating algorithm which starts with two-byte characters as tokens and based on the frequency of n-gram token-pairs, it includes additional, longer tokens. For eg, if e and r are frequently together in the language, a new token, er is added to the vocabulary. This helps encoding rare words with appropriate subword tokens without introducing unknowns. The emojis are mapped to '¡unk¿' which is mapped to 'O', hence not impacting our output.

5 Learning (8pt)

Transfer learning mechanism is used, which focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. BERTweet, trained on unsupervised tweets, is utilized for NER recognition on tweets by providing it a labeled set. As BERTweet is based on RoBERTa, it only uses masked language modeling(MLM) objective which is finetunes by using the Adam optimizer. The MLM training involves randomly hiding 15% of words in the sentence and predicting them using the hidden word context, enabling the model to understand the inner representation of the language. The Adam optimizer works with first moment,i.e mean and second moment, i.e uncentered variance of the gradients.This type of optimizer helps in rectifying vanishing learning rate and high variance. Unknown words handling isnt needed during learning as they are taken care in the tokenization step using BPE as explained earlier.

6 Implementation Details (3pt)

The implementation involves finetuning the BERTweet model. The hyperparameters involved are number of epochs, batch size, maximum length of tokens per batch. The optimizer used was Adam with parameters Adam's epsilon numerical stability and learning rate. A linear schedular was used having the parameter as number of warmup steps. To clip the exploding gradients another parameter involved was the max norm of the gradients.

7 Experimental Setup (4pt)

The entire data was used for all experimentation. I majorly tried different model architectures and number of training epochs. To keep the iterations quick, I first iterated with the other hyperparameters by keeping the epochs to 5. Once I found an architecture that worked well, I ablated with epochs. I used GPU with batching which helped speeding up the iterations. The dev F1 score was used as an evaluation metric to decide the best model.

8 Results

Test Results (3pt) The test results obtained on the leaderboard are F1 score: 0.67461, precision: 0.718 and recall: 0.636 using BerTweet model: vinai/bertweet-base and tokenizer with number of epochs: 21, batch size: 32, maximum length of tokens: 62, Adam's epsilon numerical stability: 1e-8, learning rate: 3e-5, number of warmup steps: 0 and max norm: 1.

Development Results (6pt) The dev results are present in Table 2. Started with MLP for

Figure 2: Learning Curves

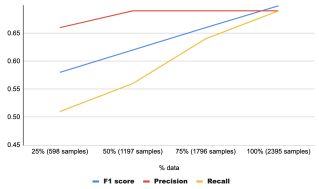
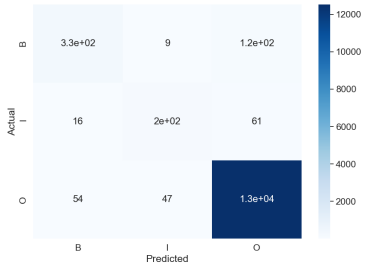


Figure 3: Heatmap of confusion matrix



baseline. Then went on ablate with BERT model architectures on learning rate. Later tried BertTweet which proved to be better. I observed that improving the number of epochs improved results significantly. I also tried techniques for inference where I stripped the sentence of emojis to get the predictions, but saw performance drop. Hence went ahead with intial inference of keeping the text as it is.

System	Prec.	Recall	F1
Development Results			
MLP(LR:1e-3, hid:64, win:2, n:3)	0.236	0.131	0.16
BERT(LR:5e-5, ϵ:1e-8, b:32, n:5)	0.4	0.56	0.47
BERT(LR:3e-5, ϵ:1e-8, b:32, n:5)	0.52	0.46	0.488
BERTweet(LR:3e-5, ϵ:1e-8, b:32, n:5)	0.652	0.685	0.62
BERTweet(LR:3e-5, ϵ:1e-8, b:32, n:10)	0.675	0.626	0.65
BERTweet(LR:3e-5, ϵ:1e-8, b:32, n:15)	0.693	0.648	0.67
BERTweet(LR:3e-5, ϵ:1e-8, b:32, n:21)	0.733	0.694	0.713
Above model in type 2 inference	0.723	0.694	0.703
Test Results			
MLP(LR:1e-3, hid:64, win:2, n:3)	0.24	0.13	0.17
BERT(LR:5e-5, ϵ:1e-8, b:32, n:5)	0.40	0.57	0.474
BERTweet(LR:3e-5, ϵ:1e-8, b:32, n:21)	0.718	0.636	0.674
Above model in type 2 inference	0.70	0.63	0.665

Table 2: Scores for various models Prec: Precision; F1: F1 score; n: epochs; LR: Learning rate; hid: hidden layers; win: window size; b: batch size; ϵ: adam epsilon

Learning Curves (3pt) As data size increases, the metrics improve.The precision stabilizes earlier although the recall continues to improve hence improving the F1 score. Also the slope goes on decreasing indicating more data beyond a certain amount wont be very beneficial.

Speed Analysis (3pt) It takes 10 seconds to get the predictions on the entire dev set and on an average of 10 ms per example. It takes 25 seconds on entire test set and 26 seconds per example. GPU hardware is used using Tesla T4 config memory size 16GB.

9 Analysis

Error Analysis (6pt) The model fails to predict names if written in lower case, such as in eg '*ram* is well,*ram*', '*jenell* i remember you', 'miss you &*lylas*'. This can be improved by preprocessing names and using a ¡name¿ token to reduce dependency on casing. Second, hyphen in between examples prevents the part after the hyphen from being a named enitity.Eg $JIN - Gone$ detects only JIN as named entity but fails to detect *Gone*. Similarly in $SugarRay - Intoyesterday$, model detects only $SugarRay$. The separator seems to be preventing the prediction and requires stripping mechanism. Third, two lettered named entities such as GA, DC present are not detected. Words with multiple senses such as *Aquarius*, 'Gone', 'ram' also fail to be detected. This implies that model still needs to be better with context understanding. There is no issue with unknown words.

Confusion Matrix (3pt) The most common confusion occurs in detecting the start token 'B' which is mostly predicted as 'O'. 'O' is predicted correctly as that is anyways the default choice. It is observed that most of the words which have dual meanings are not detected as entities such as 'ram', 'Gone' present in the dev set leaving them undetected. While most of the common entities such as 'Brazil', 'beyonce' were correctly detected seemingly due to the model being pretrained on lot of data. The undetected examples seem hard because of its rarity in occuring as a named entity.

10 Conclusion (2pt)

Transfer learning using BERT-based architectures prove to be highly effective in attaining high F1 scores of 0.713 on dev set and 0.674 on test set despite using very small labeled dataset. The training can be made more powerful by adding extra processing such as name detector to make names capitalized and converting acronyms of known nouns to full text to improve results.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. 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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[2] 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, 2019.

[3] Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. Bertweet: A pre-trained language model for english tweets, 2020.

[4] Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank J Reddi, Sanjiv Kumar, and Suvrit Sra. Why {adam} beats {sgd} for attention models, 2020.