CS5740: Project X
https://github.com/cornell-cs5740-20sp/final-adamwlev

Adam Levin
awl92

1 Introduction (6pt)

In this paper we detail our named entity recognition system which is trained and evaluated on annotated Tweets from Twitter.

Our system uses state of the art NLP building-blocks - a pre-trained RoBERTa transformer model from Hugging Face Transformers (HFT) and Long Short-Term Memory Neural Network - to achieve very accurate results while maintaining the requisite speed at inference time. This system allows us to adapt pre-trained neural networks (on large corpuses) to the specific task.

We experimented with many different models and decoders such as BERT, LSTM and CRF. We also varied the amount of data, the size of the transformer model, and the hyperparameters of our learning process.

2 Task (5pt)

The task we work on is Named Entity Recognition. The goal is to correctly identify spans within sentences which represent named entities such as people, place, business, etc. without having to specify the type of named entity. Formally, the task is defined as follows: we would like to label a sequence $\overline{x} = \langle x_1, ..., x_n \rangle$ as a sequence of tags $\overline{y} = \langle y_1, ..., y_n \rangle$ with the tags defined by $\mathcal{Y} = \{\text{“}O\text{”}, \text{“}B\text{”}, \text{“}I\text{”}\}$, with “O” being not an entity, “B” being the beginning of an entity and “I” being the continuation of an entity.

3 Data (4pt)

Our technique uses the data in raw form. We use a sub-token tokenizer (the same as was used in the pre-trained transformed model) which eliminates the need to have special treatment for unknown tokens and the need to preprocess data. When we tried preprocessing such as lowercasing of the tokens, performance degraded. Additional training data was found which seems to belong to the same dataset as the one provided except with more training examples.[1] A script was used to take the subset of the additional data examples that were not present in the original train, dev and test sets to ensure no duplicates or cheating with regard to the leaderboard evalauation (some of the test examples were found annotated in the additional dataset).[2] The additional training data improved performance (see Experiment Sections). See Table 1 for statistics on the data.

Statistic	**Training**	**Dev**	**Test**
# Tweets	5676	959	2377
# Tokens (thousands)	101.6	13.4	33.3
Longest Tweet	101	41	52
# Unique Tokens	20,773	5,168	10,548
% Tokens in Trainset	100	76.3	76.5

Table 1: Data Stats. No preprocessing performed.

4 Model (25pt)

Our model starts by transforming each token of a Tweet to subtokens using the pre-bult RoBERTa tokenizer from HFT. The model then produces a contextually-aware embedding by running the subtokens through the RoBERTa transformer. It then performs an averaging of all subtoken embeddings for each whole token so that we obtain one embedding per token in the tweet. This embedding is then passed to an LSTM model (trained from random initialization) which is depicted in a rolled out fashion in Figure 1. Each hidden state which comes out of the LSTM model is then classified using a single Linear layer to transform it to the logit predictions of each class. In equations,

[1] https://github.com/juand-r/entity-recognition-datasets/tree/master/data/WNUT17

[2] https://github.com/cornell-cs5740-20sp/final-adamwlev/blob/master/create_wnut_ds.py

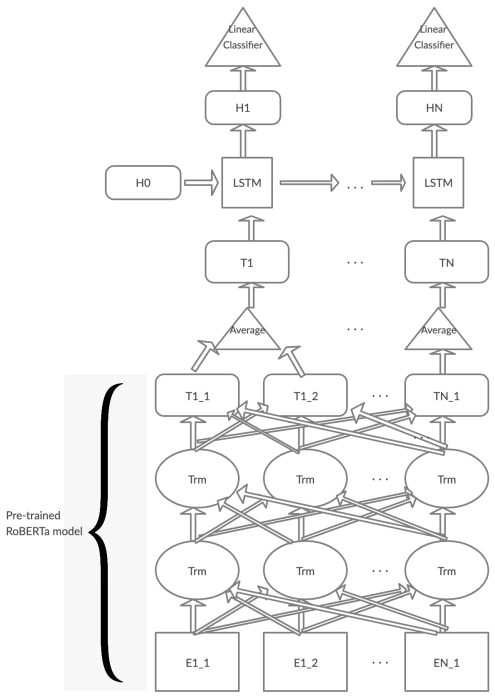

Figure 1: Diagram of our model, BertTagger.

our model is:

$$\begin{aligned} t_s &= \text{RoBERTA}(e_s) \\ t_t &= Rt_s/\text{ColumnSum}(R) \\ h_t &= \text{LSTM}(t_t) \\ o_t &= Lh_t \end{aligned}$$

with all s subscripts meaning subtoken vectors and all t subscripts meaning token vectors. RoBERTa is the unpooled pre-built transformer module from HFT. We use the RoBERTa large with 12 layers, hidden dimension 1024 and 16 attention heads. R is a matrix which maps from the subtoken space to the token space and has dimensions MAX_NUMBER_TOKENS by MAX_NUMBER_SUBTOKENS (with each being constants determined at runtime by the input data). L is a linear layer which maps from Hidden state of the LSTM to the class space.

During inference, we simply do a forward pass of the model, the same that we do during training. Additionally, since our model uses sub-token parsing, there is very little issue with unknown words. Occasionally in the data there are foreign characters such as emojis that are unknown to the pre-built tokenizer for which the tokenizer automatically uses a special UNK token.

5 Learning (10pt)

We use Cross-Entropy Loss, a standard classification loss as our objective function. Our learning is performed by differentiating our objective with respect to our learn parameters (including the RoBERTa parameters, initialized to pre-trained weights) and updating them via backpropagation. Formally our objective is:

$$L(w) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

As mentioned in the Model section, unknown words are not an issue in our model since we use a pre-trained sub-token tokenizer. Therefore we do not have any parameters associated with unknown or rare words in our learning procedure. We train our model until the F1-score of the prediction spans on a validation set stops improving.

6 Implementation Details (3pt)

We use the AdamW optimizer which performs Adam optimization and simultaneous regularization of the L2 Norm of the weights. We use a Learning Rate of 8e-6 on all our parameters and an L2 penalty coefficient of 1e-3. Since our model is large, we increase the effective batch size by accumulating gradients over multiple backward passes before taking an optimization step. We use an effective batch size of 90 Tweets.

7 Experimental Setup (4pt)

For all our experiments we use the RoBERTa Base size (for reasonableness of computation), we train until performance on a validation set has stopped improving. We report results using F1 score on the Development set unless otherwise noted.

8 Results

Test Results (3pt) We obtained a F1-Score of 0.697 on the leaderboard with a precision of 0.723 and a recall of 0.672. We used our best setup (described in Model and Implementation Details) to generate this result. We also trained on all available annotated data (original training data, development data, supplemental training data) to generate this result after determining the ideal number of epochs to train for (9) by monitoring performance on a validation set while training.

Development Results (7pt) Our development ablations are displayed in Table 2. We show results with and without our LSTM decoder, with the Base and Larger version of our the pre-trained

Development Results	F1-Score
RoBERTa-Base + Linear	0.6913
RoBERTa-Base + LSTM + Linear	0.6879
RoBERTa-Large + Linear	0.7226
RoBERTa-Large + LSTM + Linear	0.7402
RoBERTa-Large + LSTM + Linear without supplemental training data	0.4225

Table 2: Development Results.

transformer model, and with and without the supplemental training data (see Data section for details). The decoder clearly helped a bit and so did the large version of the transformer. However, what helped by far and away the most was the increase in amount of training data.

Learning Curves (4pt) The Learning Curve (displayed in Figure 2) clearly shows the value of more training data for our system. If we had not used our supplemental training data (red dashed line in plot) our performance would have been much worse.

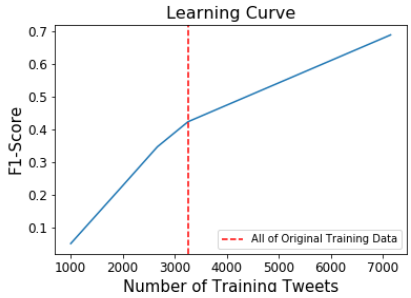

Figure 2: Learning Curve.

Speed Analysis (4pt) Our algorithm processes 22.6 Tweets per second and 317.7 Tokens per second. To compute this metric we had our model on a GPU on Google Collab and we ran through all the test data set examples 1 by 1 (no batching so that it can simulate real use for example in an API).

9 Analysis

Error Analysis (7pt) Two classes of errors that we found were when named entities share the same word with non-named entities and when annotations were oddly inconsistent in the data. Both classes of error are difficult to alleviate. The second class is out of our control and the first class is difficult to overcome when using the pre-trained transformers model. Please see Table 3 for examples of these errors. One case of an unknown word causing an error was when an emoji was in a named entity. Our model does character level embeddings but did not have an embedding for an emoji.

Tweet	Named Entities
Words That Are Also Non-Entities	
"RT @basedram: And sober ram is well, ram"	"ram","ram"
"@ChelsTurnerrr I already told chase I want to do something tomorrow!!"	"chase"
"Some good results so far today always a happy bunny when United win & Arsenal lose. Well done spurs now #COYI"	"United", "Arsenal", "spurs"
Odd Annotations	
"@ROBOtussinn lol what's with the face.., I'm for real"	"face"
"#NP: JIN - Gone."	"JIN","Gone"
"Hot New Music: Rae Sremmurd ft Nicki Minaj & Pusha T - No Flex Zone (Remix) — http://t.co/Fl5UL3zSQu"	"Pusha T - No Flex Zone"

Table 3: Two classes of common errors (missed named entities in development set).

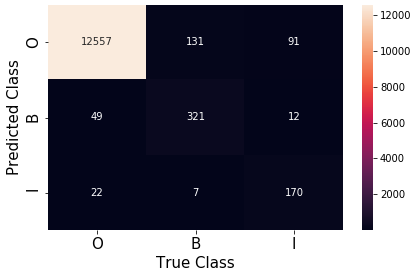

Figure 3: Confusion Matrix.

Confusion Matrix (5pt) Our Confusion Matrix (Figure 3) shows the difficulty of recall for "B" and "I" tags with this task. Since there are many named entities which are unknown words and some of these words can exist is non named entities, it is difficult to have good recall on this task without very low precision. An improvement might be to try a weighted objective function to penalize these missed named entities more.

10 Conclusion (3pt)

In the end we managed to achieve respectable results on this difficult dataset. There were many challenges such as messy data and questionable annotations. What worked was more training data and pre-trained transformer models trained on sub-word tokens to minimize issues with unknown words. In terms of the leaderboard, a jump from 45th place to 4th place was achieved by simply finding additional training data (assuming development set has similar results as test set).