CS5740: Project X

https://github.com/cornell-cs5740-20sp/final-shivsi

Ananya Shivaditya
aks298

1 Introduction (6pt)

In this report, we implemented the task of Named Entity Recognition (NER). Given a sentence, label each word as B (Beginning of NE), I (Continuation of NE), or O (not NE). This task is to be optimised for accuracy and speed of inference.

In order to do so, this report explored the implementations with combinations of Bidirectional Long Short Term Memory (LSTM) and Conditional Random Field (CRF) units in a Recurrent Neural Network (RNN) model. 50-dim GloVe pre-trained word embeddings were used to augment contextual information, alongside the choices of model architecture, given constraints of not using pretrained models on larger datasets (for the same specific task of NER).

The main experiments conducted in this report include exploration of architectural units in the RNN [LSTM, GRU] as well as size of dataset. The metrics included accuracy and time taken for inference. In addition to the provided dataset, this report makes use of kaggle dataset[1]. The best F1 score achieved on the test set[2] was 0.126, with precision of 0.08, and recall of 0.307. This result was achieved with a bidirectional LSTM model, with hidden dimension of 50.

2 Task (5pt)

Let $\mathcal{V}$ be the set of all possible natural language words and $\mathcal{Y}$ be the set of Classification tags {B, I, O}. $y \in \mathcal{Y}$ is a classification tag. A sequence of tokens $\bar{x}$ composed of individual tokens $\langle x_1, \ldots, x_n \rangle$ of length n is to be mapped to a sequence of classification labels with same length n i.e. $\bar{y} = \langle y_1, \ldots, y_n \rangle$. This report aims to define this function $f(\bar{x}) = \bar{y}$ as accurately as possible, as well as constrain the time taken to compute $\bar{y}$ in order to aid politicians on Twitter to quickly recognise the entities that they may refer to in their Tweets.

Output space $\mathcal{Y}$ consists of 3 tags below:

[1] entity-annotated-corpus from Kaggle

[2] Previous leaderboard run on 1 PM, April 20, 2020

B	Named Entity start	eg. First Name
I	Named Entity continued	eg. Last Name
O	Not a named entity	eg. common noun, verb

3 Data (4pt)

Data used consists of both provided data (twitter stream) and Kaggle dataset[1]. Descriptions of each are tabulated below, along with how the characteristics were computed.

1. Number of sentences in training set
2. Number of sentences in development set
3. Number of sentences in test set
4. Vocabulary size

	Kaggle	Twitter	Total
1.	33570	2163	35733
2.	7194	960	8154
3.	7194	2377	9571
4.	35180	20343	49930

Size of $\mathcal{V}$ was computed as the number of unique words within the dataset. Total vocabulary size is not equal to the sum of dataset vocabularies because of words found in both sets. Each word was parsed with a regular expression to only allow alphabet and punctuation. It was then lemmatized using the NLTK's Word_Net_Lemmatizer and readied further as its 50-dim GloVe vector embedding. Unknown words were marked with the <UNK> symbol and were embedded separately. These tokens were concatenated in a sentence and loaded in batches. Each sentence was padded with pad_tag = -1 to equal the maximum length of a sentence in the batch.

4 Model (25pt)

A single bidirectional RNN with a long short-term memory (LSTM) recurrence was used to encode the sentence. Sequences of tokens $\bar{x}$ are mapped to $\mathcal{Y}$ tags by iterating through each sentence and generating the

y for each token. The execution includes generating the context available due to all previous steps at each step. In the case of a bidirectional LSTM unit, there is also context from the tokens ahead of the current step in the sentence. This adds value to the final output layer, which is is the final step before prediction of a y. The tag context $\widetilde{s}_k$ helps decide policy for selecting a y is modeled by the function $\Pi_\theta(\widetilde{s}, y)$. This is a probabilistic neural network parametrized by θ, where $\widetilde{s}$ is the tag context at step k and y is a tag class.

The forward LSTM RNN hidden states are computed as

$$\mathbf{h}_{j+1} = LSTM^E(\phi^I(x'_{j+1}); \mathbf{h}_j)$$

where ϕ^I is the learned word embedding function and $LSTM^E$ is the forward LSTM recurrence function. We use a similar computation to compute the backward hidden states $\mathbf{h}_j$. For each token x'_j in $\bar{x}'$ a vector representation $\mathbf{h}_j$ is computed.

To generate executions, we generate one y at a time, compute the new context and observe the new $\widetilde{s}$ state. A single bidirectional RNN with a long short-term memory (LSTM[3] recurrence was used to encode the sentence. Contexts are generated at each step by the biLSTM model and fed to the last softmax layer to give probabalistic outputs for each token. The token withe highest probability is chosen and the next batch of sentences is loaded for evaluation.

Unknown words were tokenized if they occur with a frequency less than a cutoff (default = 5) in our training data set and training the model using the mapped tokens instead. The embedding of these mapped tokens are calculated the same way as the unmapped words by the embedding function. By mapping the rare words to a singular new token, we have increased the number of occurrences in the training data for this new token and can learn more about the probability of it being mapped to an output tag. The cutoff parameter is dependent on the training data.

5 Learning (10pt)

We estimate the policy parameters θ using an exploration based learning algorithm that maximizes the immediate expected reward. Broadly speaking, during learning, we observe the token output classification given the current θ, and for each visited state compute the expected immediate reward by observing rewards for all output classes of tags. This is done by calculating the error as cross entropy loss in terms of incorrect tags, and decreasing the probability of a similar mis classification in the future. The loss function used is log cross entropy loss, which helped by gradient of the function updates weights by factor contributed by each parameter in the network.

$$L = -log(\frac{\Sigma_i^{|\bar{x}|}(y == f(\bar{x}))}{|\bar{x}|})$$

In order to update the weights of the network, each node's weight is affect in every training epoch by factor of it's gradient to the final layer. This gradient is calculated internally by the pytorch framework. The Adam optimizer was used to navigate the hyperplane of the loss function and include momentum (memory of previous direction of descent towards global optimum).

The LSTM model was better trained on the additional Kaggle dataset, which had a smaller vocabulary than from Twitter, and almost just as many sentence examples, which were more structurally sound.

Handling of unknown words was described earlier as parameterising the cutoff value for occurrance in the training data, and mapping those rare words to a new token to be learned with.

6 Implementation Details (3pt)

A single bidirectional RNN with a long short-term memory (LSTM) recurrence was used to encode the sentence, and then were fed to the network in batches of size 10, and padding with -1 on the end for sentences shorter than the maximum length of the sentences in that batch. The number of hidden states in the bidirectional LSTM was set at 50. The dimension of the GloVe embeddings of each token was 50. The output of this layer was gathered as a contiguous sequence and fed to a fully connected layer of dimension 100. This was then taken forward to the last layer by applying log softmax on each token's output since it is numerically more stable in computation. The Adam optimzer was used, with learning rate of 1e-4 for 20 epochs, 9e-5 for 10 epochs and 8e-5 for 15 epochs. This was done to achieve global minimum of the loss function. The GPU architecture on Google Colab enabled a batch size of 10 for reasonable train time of approximately 30 mins.

7 Experimental Setup (4pt)

The data used in experiments refers to the total data, combination of both the provided Twitter dataset and aforementioned Kaggle dataset. The systems for comparison included 1. Amount of data used

[3]Hochreiter and Schmidhuber, 1997

for training 2. Hidden dimension of biLSTM unit 3. Learning rate for training. Evaluation was done for accuracy on development set and peed of inference of each sentence $\bar{x}$.

8 Results

Test Results (3pt) Experimental results as seen in the Table 1 the best F1 score on the dev dataset was 0.967117 (line 4.) This was achieved using the bidirectional LSTM model with hidden dimension 50 and learning rate = 1e-2. It made use of the Kaggle data for training. This model achieved on the test set on the leaderboard [2] was 0.126, with precision of 0.08, and recall of 0.307.

Development Results (7pt) Table 1 shows results of accuracy and speed on the development set. It includes ablations for variations in dataset (size and source) and learning rate during training. The hidden dimension of the biLSTM model is also tested for optimization of accuracy. The dimension of the GloVe embeddings of each token was 50, and was not altered across the paramenter space. The optimizer Adam was used, it was noticed that its performance far quickened the training process, with less than 20 epochs for convergence. Compared to the AdaGrad optimizer, this was much faster: Adagrad took nearly 100 epochs for convergence.

From the table, we notice that the highest accuracy is obtained by the LSTM model with learning rate of 1e-2 (line 4). This value was arrived at by a hyperparameter grid search after varying the other parameters as well.

We also notice that accuracy increases by a slight margin with increase in the hdden dimension of the LSTM model (lines 7-9). This is expected, since a larger hidden dimesension would capture a higher degree of context from a sentence, therefore increasing the complexity and power of the model. Also, this does not have a cost in terms of time cost of inference, as we see that the time cost varies only slightly across the dimensional ablations.

Accuracy in terms of dataset size and quality is discussed below.

Learning Curves (4pt) As seen in the table, we benefit with a limited portion of the data (lines 1 and 2) to a higher degree than with the total data (line 3.) It is further seen that the Kaggle dataset is more useful than the Twitter data. This is expected, since the Twitter data contained ungrammatical sentences, non-words (such as emojis and slang words) as well as symbols like hashtag. The LSTM model was better trained on the Kaggle dataset, which had a smaller vocabulary than from Twitter, and almost just as many sentence examples, which were more structurally sound. However, these results did not perform well on the leaderboard, since the test data was also from Twitter feeds.

Speed Analysis (4pt) As seen in Table 1, the speed of inference from model variations was not severely affected. These speeds were calculated as the mean of time taken for inference within each batch of evaluation, and then averaged across all batches of the validation data. The model was trained on Google Colab's GPU architecture, with 12 GB of RAM.

It is further noticable that the length of each sentence becomes a significant factor for inference. This can be inferred from the max length of the sentence in the Kaggle dataset was 60 (speed = 960), whereas in the Twitter dataset it was 40 (speed = 800), which reflects in the speed. This tallies with the need to compute contexts for each sentence in a batch, and since each sentence in a batch is padded till the maximum length of its sentences, the effect is compounded during inference (and training.)

9 Analysis

Error Analysis (7pt) Error classes include B and I, for their sparsity in the training data. Also, tokens like emojis and symbols or slangs were heavily misclassified, such as "futballfans' Unknown words were treated with a cutoff, and lacked contextual clarity, and were definitely difficult to tag.

Confusion Matrix (5pt) It was especially difficult to classify tag of B and I, since there existed tokens with similar contexts and begining just one token later, labelled incorrectly as a continuation (I). It was difficult to classify using BERT since the output required each token to be classified, with prior tokenization. BERT embeddings require to be tokenized given the entire sentence.

10 Conclusion (3pt)

This report concludes that the biLSTM model with hidden dim 50 captures context in word embeddings and can effectively produce inference to tag words as Named Entities.

	Experiment	Accuracy	Loss	Speed
1.	data: Twitter	0.947400	0.159764	963.2
2.	data: Kaggle	0.963094	0.103195	800.7
3.	data: Total	0.960606	0.113424	798.5
4.	LR: 1e-2	0.967117	0.10166	765.2
5.	LR: 1e-3	0.966924	0.100629	792
6.	LR: 1e-4	0.960624	0.112981	763.1
7.	h. dim: 50	0.960512	0.112163	803.6
8.	h. dim: 70	0.961845	0.109057	801.6
9.	h. dim: 80	0.962108	0.10905	802.5

Table 1: Experiments on dataset, speed of inference(samples/second), learning rate (LR) and hidden dimension of LSTM model (h. dim)

Citations

1. Entity-annotated-corpus from `https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus`

2. Leaderboard run link on 1 PM, April 20, 2020 `https://github.com/cornell-cs5740-20sp/leaderboards/blob/4025003dd290f935f5a61652a6d7d1870eb7457c/final/leaderboard.csv`

3. Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural computation, 9.