Event Detection Task Using Deep Learning

Sameer Ahmed
Sapienza University of Rome
Multilingual Natural Language Processing

Abstract

Event detection (ED) is a task in Natural Language Processing (NLP) that aims to identify event triggers and categorize events mentioned in the text. Event detection is essential because it may be used to automatically annotate more linguistic resources and improve the precision of other NLP tasks that depend on event detection, including question answering or machine translation. In this paper, the author presents different deep learning approaches with a remarkable F-1 score to find out event detection in the pre-tokenized sentence, these event detection labels are based on the BIO format. The total number of labels according to BIO format is 11 (B-Sentiment, B-Scenario, B-Change, B-Possession, B-Action, I-Sentiment, I-Scenario, I-Change, I-Possession, I-Action, and O).

1 Introduction

An event is defined as something that takes place at a certain time and location. It could be a tragic human-made event like an accidental death or a natural disaster like an earthquake or flood. For example, "Previously, Bohol was also hit by an earthquake on February 8, 1990, that damaged several buildings and caused a tsunami." This example describes the natural disaster "earthquake", It is common nowadays that this type of information we get from electronic newspapers or informative platforms like 'Wikipedia'. But due to the rapidly increasing number of news articles that are being published daily, it's impossible to extract events manually and understand what the event is about. This problem is solved by event detection (ED) in an easy and very effective way by using a deep learning approach.

2 Approach

The flow of this task is shown in Figure 1. Where the inputs are lists of tokens that make up the sentence. And our task is to find out event detection with their labels.

3 Data Preparation

Our dataset is not in the form that we give to our model. We need to preprocess our training, testing and development (Validation) data.

1. The first step is to convert (training, development, and testing) tokens into lowercase because the capitalized forms of the words will give different embeddings from the lowercased forms.
2. Then convert (training, testing, and development) tokens and labels into numbers. These tokens and labels need to be converted by taking training data (tokens and labels) unique values.
3. All tokens and labels are not the same length. So, for that, we need to apply padding.

4 Model Architecture

It is a sequence labeling task, and for sequential data, the best approach is to use RNN. This task is 'many to many', where our input also consists of many words, and we need to compute many labels as our output. To solve this problem, I am trying 3 different types of models.

4.1 Bidirectional LSTM (BiLSTM) without pretrained word embedding

The first approach of the model is Bidirectional LSTM (BiLSTM) without pre-trained word embedding, the architecture of the model is shown in Figure 2. BiLSTM may use data from both sides, unlike LSTM, the input flows in both directions of the sequence, it is a powerful tool for modeling the sequential relationships between words and phrases. Before this BiLSTM layer, I also added a word embedding layer, With the use of word embeddings, It helps the model to figure out relationships in language. I tried different hyperparameters to increase the model F-1 score. like change activation function, optimizer, embedding dimension, hidden dimension of BiLSTM layer, Linear layer, etc. but the F-1 score did not increase with the combination of different hyperparameters. So, I finalized the architecture of the model shown in Figure 2. I tried categorical cross-entropy loss with ignore index (for padding) because the task is to predict 11 different classes.

4.2 Bidirectional LSTM (BiLSTM) with pre-trained word embedding

. In this model, I am using Bidirectional LSTM (BiLSTM) with pre-trained word embedding. Here I am using a pre-trained word embedding layer instead of an embedding layer. The pre-trained embedding layer will increase model F-1 score because it's trained in a bunch of vocabs already. So, it will work better. The first thing is to create a word embedding matrix for those words, which is available in my training dataset. This thing takes less computation power because we don't need every word from pre-trained, we only need the vocabulary that is available in my training dataset. Here I am using a pre-trained word embedding as 'glove-wiki-gigaword-300'. It's basically 300-dimensional word embedding based on 400k different vocabulary. I tried different hyperparameters like a change of pre-trained word embedding, activation function, etc. But the F-1 score is decreasing instead of increasing. So I finalized the architecture of the model shown in Figure 3. I tried categorical cross-entropy loss with ignore index (for padding) because the task is to predict 11 different classes.

4.3 Bidirectional LSTM with Conditional random field (CRF)

In the Bi-LSTM model, we get logits of every class for each word. Prediction of the Bi-LSTM of the word is not dependent on the prediction of its neighbor word. For example, if you see Fig 4 (a) and 4 (b). The word "damage" can be different entities depending on the context. In Fig 4 (a) the word damage label is "B-Change" and in Fig 4 (b) word damage label is "O". So, if we need the information on neighbor words in the sense of context, Bi-LSTM fails over there. The solution is to use a CRF layer. CRFs are a sort of probabilistic graph model that considers the context of nearby sample data. In this approach, I am using the CRF model with the Bi-LSTM layer because the probability distribution and emission score matrix are given to CRF as input through BiLSTM, which enables CRF to learn from the input how to label the sequence. This allows CRF to learn its internal logic.

5 Result

For evaluation, we are not calculating accuracy as an evaluation metric, but macro F1-score is best for this task. Because most of the labels are "O", shown in Figure 5. The 1st model Bidirectional LSTM (BiLSTM) without pre-trained word embedding showed decent results up to the 30th epoch, the model results are shown in Figure 6(a), 6(b), and 6(c). The model results are quite good but not enough. So, I tried the 2nd model Bidirectional LSTM (BiLSTM) with pre-trained word embedding, it worked better compared to the previous model as shown in Figures 7(a), 7(b), and 7(c). but the macro F-1 score was under 50. Now this time, I tried BiLSTM with the CRF model, and the results are higher compared to the previous models only in 2 epochs as shown in Figures 8(a), 8(b), and 8(c). Then I decided to resume the training for 1 epoch, but the decision is not fruitful because the F-1 score was decreased instead of increased, as shown in Figures 9(a), 9(b), and 9(c). So, it means our 3rd model (BiLSTM with CRF) has the best results, compare to the other 2 models, If we train on 2 epochs.

6 Conclusion

In this paper, I systematically compared 3 different models for event detection (ED) sequence labeling task. Shown in Table 1. BiLSTM with the CRF model worked better compared to the other 2 models because it's less dependent on word embedding.

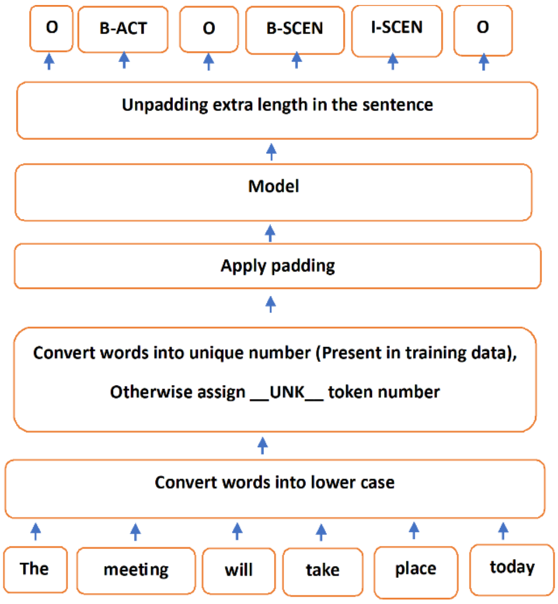

Figure 1: Flow diagram of model approach

```
model_1 = MODEL_1(hidden_dim=hidden_dim, output_dim=output_dim,embedding_dim=embedding_dim,dropout_value=dropout_value).to(device
optimizer = optim.Adam(model_1.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss(ignore_index=tags_tokenizer['__PADDING__'])

model_1

MODEL_1(
  (embedding_layer): Embedding(33081, 300, padding_idx=0)
  (lstm): LSTM(300, 128, batch_first=True, bidirectional=True)
  (dropout1): Dropout(p=0.4, inplace=False)
  (fc1): Linear(in_features=256, out_features=32, bias=True)
  (dropout2): Dropout(p=0.4, inplace=False)
  (fc2): Linear(in_features=32, out_features=12, bias=True)
  (softmax): Softmax(dim=1)
)
```

Figure 2: Bidirectional LSTM (BiLSTM) without pretrained word embedding

```
model_2 = MODEL_2(hidden_dim=hidden_dim, output_dim=output_dim,embedding_dim=embedding_dim,dropout_value=dropout_value,weight=wei
optimizer = optim.Adam(model_2.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss(ignore_index=0)

model_2

MODEL_2(
  (embedding_layer): Embedding(33081, 300, padding_idx=0)
  (lstm): LSTM(300, 128, batch_first=True, bidirectional=True)
  (dropout1): Dropout(p=0.4, inplace=False)
  (fc1): Linear(in_features=256, out_features=32, bias=True)
  (dropout2): Dropout(p=0.4, inplace=False)
  (fc2): Linear(in_features=32, out_features=12, bias=True)
  (softmax): Softmax(dim=1)
)
```

Figure 3: Bidirectional LSTM (BiLSTM) with pretrained word embedding

{"idx": 18494,
"tokens": ["The", "``", "Luftwaffe", "''", "formations", "were", "dispersed", "by", "a", "large", "cloud", "base", "and", "failed", "to", "inflict", "severe", "damage", "on", "the", "city", "of", "London", "."],
"labels": ["O", "O", "O", "O", "O", "O", "O", "B-CHANGE", "O", "O", "O", "O", "O", "O", "O", "O", "O", "B-CHANGE", "O", "O", "O", "O", "O", "O"]
}

Figure 4 (a): The word ‘damage’ in different context

{"idx": 18461,
"tokens": ["The", "storm", "was", "responsible", "for", "one", "death", "and", "$", "100,000", "in", "damage", ",", "mostly", "in", "North", "Carolina", "."],
"labels": ["O", "O", "O", "O", "O", "O", "O", "O", "O", "O", "O", "O", "O", "O", "O", "O", "O", "O"]}

Figure 4 (b): The word ‘damage’ in different context

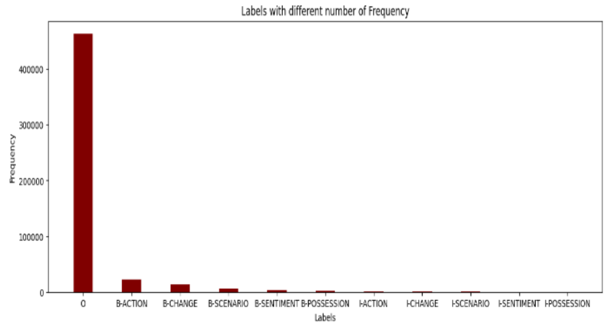

Figure 5 Labels of the model with frequency

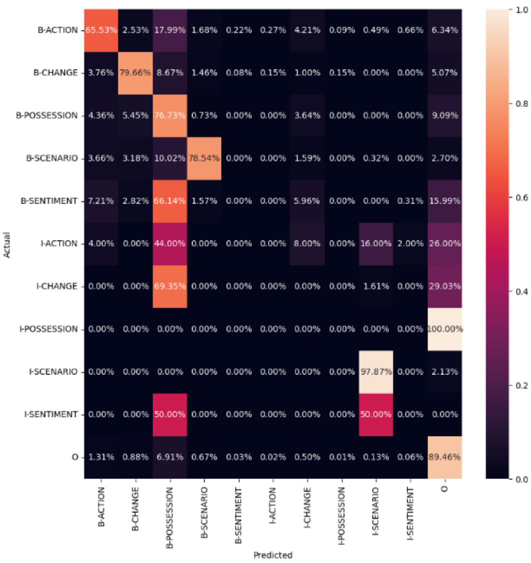

Figure 6 (a): Confusion Matrix of model 1 (Bidirectional LSTM without pretrained word embedding)

```
MODEL-1 (Scheme=IOB2
==============
              precision    recall  f1-score   support

      ACTION       0.58      0.70      0.63      2257
      CHANGE       0.62      0.74      0.68      1303
  POSSESSION       0.00      0.00      0.00       275
    SCENARIO       0.52      0.85      0.64       629
   SENTIMENT       0.04      0.01      0.02       319

   micro avg       0.57      0.64      0.61      4783
   macro avg       0.35      0.46      0.39      4783
weighted avg       0.51      0.64      0.57      4783
```

Figure 6 (b): F-1 Score of model 1 (Bidirectional LSTM without pretrained word embedding)

```
print(accuracy_score(true_tags_WOP_m1,pred_tags_WOP_m1))

0.872293116782676
```

Figure 6 (c): Accuracy of model 1 (Bidirectional LSTM without pretrained word embedding)

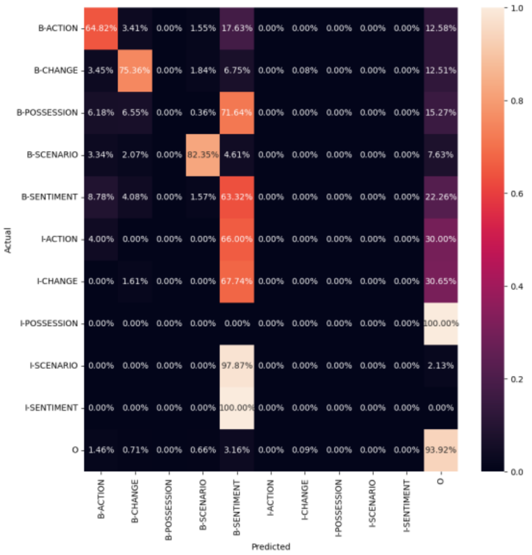

Figure 7 (a): Confusion Matrix of model 2 (Bidirectional LSTM with pretrained word embedding)

```
MODEL-2 (Scheme=IOB2
===============
              precision    recall  f1-score   support

      ACTION       0.64      0.64      0.64      2257
      CHANGE       0.65      0.72      0.68      1303
  POSSESSION       0.00      0.00      0.00       275
    SCENARIO       0.53      0.75      0.62       629
   SENTIMENT       0.08      0.63      0.14       319

   micro avg       0.43      0.64      0.51      4783
   macro avg       0.38      0.55      0.42      4783
weighted avg       0.55      0.64      0.58      4783
```

Figure 7 (b): F-1 Score of model 2 (Bidirectional LSTM with pretrained word embedding)

```
print(accuracy_score(true_tags_WOP_m2,pred_tags_WOP_m2))
0.9105761794276875
```

Figure 7 (c): Accuracy of model 2 (Bidirectional LSTM with pretrained word embedding)

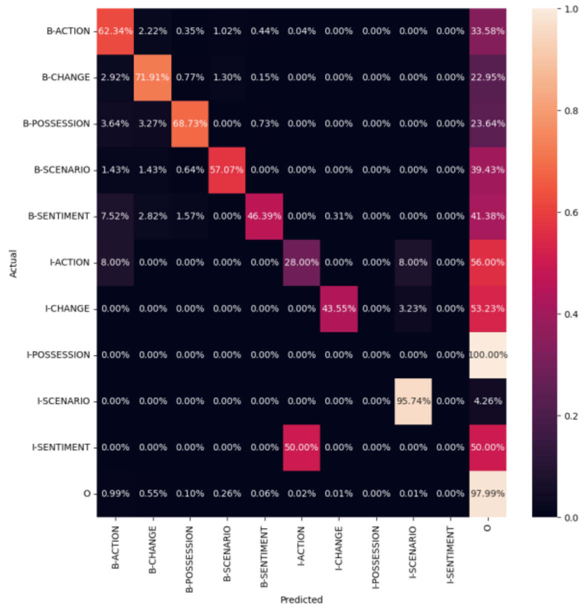

Figure 8 (a): Confusion Matrix of model 3 (Bidirectional LSTM with Conditional random field)

```
MODEL-3 (Scheme=IOB2
===============
              precision    recall  f1-score   support

      ACTION       0.71      0.62      0.66      2257
      CHANGE       0.73      0.71      0.72      1303
  POSSESSION       0.71      0.68      0.70       275
    SCENARIO       0.69      0.57      0.62       629
   SENTIMENT       0.77      0.46      0.58       319

   micro avg       0.72      0.63      0.67      4783
   macro avg       0.72      0.61      0.66      4783
weighted avg       0.72      0.63      0.67      4783
```

Figure 8 (b): F-1 Score of model 3 (Bidirectional LSTM with Conditional random field)

```
print(accuracy_score(true_tags_WOP_m3,pred_tags_WOP_m3))
0.9466744006187162
```

Figure 8 (c): Accuracy of model 3 (Bidirectional LSTM with Conditional random field)

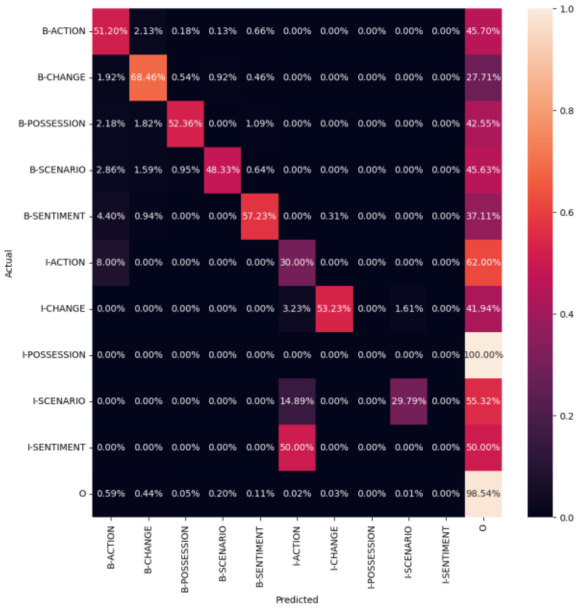

Figure 9 (a): Confusion Matrix of model 3 again train for 1 epoch (Bidirectional LSTM with Conditional random field)

```
MODEL-3
===============
              precision    recall  f1-score   support

      ACTION       0.76      0.51      0.61      2256
      CHANGE       0.76      0.68      0.72      1303
  POSSESSION       0.77      0.52      0.62       275
    SCENARIO       0.73      0.48      0.58       629
   SENTIMENT       0.70      0.58      0.64       318

   micro avg       0.75      0.56      0.64      4781
   macro avg       0.75      0.56      0.63      4781
weighted avg       0.75      0.56      0.64      4781
```

Figure 9 (b): F-1 Score of model 3 again train for 1 epoch (Bidirectional LSTM with Conditional random field)

```
print(accuracy_score(true_tags_WOP_m3,pred_tags_WOP_m3))

0.9445948559272868
```

Figure 9 (c): Accuracy of model 3 again train for 1 epoch (Bidirectional LSTM with Conditional random field)

	Epoch	**F-1 score (macro)**	**Accuracy**
BiLSTM with pretrained word embedding	30	0.39	87 .2%
BiLSTM with pretrained word embedding	25	0.42	91.0%
BiLSTM with CRF	**2**	**0.66**	**94.6%**
BiLSTM with CRF	3	0.63	94.4%

Table 1: comparison of different models for sequence labeling task