Automatic Event Extraction from news documents written in Indian languages with deep neural networks

Alapan Kuila

Indian Institute of Technology Kharagpur

1 Introduction

We are working on Event Extraction from news documents. In this work we have processed the news documents first, then extracted the relevant events from the processed news document and at the end represented the events in structured format. Our work covers the events which depicts the occurrence of any disasters caused by nature (e.g. Flood, Earthquake etc.) or manmade (e.g. terrorist attack, accident, riot etc). The term **Event** covers the type of the event and the event participants or arguments which depicts the informations regarding that event(i.e. when[Time], where[Place], cause, effect etc). So while extracting events we have to identify the type of the extracted event as well as the event arguments related to that event. Most of the existing event extraction systems rely on Event trigger identification and get knowledge about the event by identifying the trigger word and the type of the trigger. Trigger words are the word or phrase that most clearly indicate the occurrence of an event. Then this trigger words are mapped with the event attributes(time, place, cause, effect etc.) which are extracted by an argument identifier to get the arguments related to the event.

We have used the same procedure and identified the trigger words present inside the document. Our neural network based model which is a combination of Convolution Neural Network(CNN) and Recurrent neural network (RNN), takes each word and its context as input and decide what is the corresponding event trigger type of that word. The trigger type is *None* if it is not a valid event trigger. We have classified disaster type events in total 32 sub-categories. The overall precision and recall of our trigger classification system is 0.46 and 0.39. After that we have applied an argument identifier which will identify the attributes which will be relevant to the event triggers. THe overall accuracy of our argument extraction model is precission:0.60 and recall: 0.50. After identifying the event triggers and arguments we have to link the arguments to their corresponding evetn trigger. From the news documents it is noticed that not all the arguments of a specific trigger present in the same sentence in which the event-trigger exists. The arguments are distributed all over the document. And we have to link all those event-arguments to get an document level idea of the specific event. So to link the trigger-argument pair we have taken a huristic based approach where for each argument phrase we will find the nearest trigger word. And will link that trigger word with that argument phrase. Thus we will get the argument phrases for each event trigger.

2 Methodology

As we have no such domain knowledge of languages like Hindi and Tamil, so we have designed our model more or less language independent. We formalize the Event Detection(ED) problem as a multi-class classification problem via combination of convolution neural network(CNN) and bidirectional Long short term memory (Bi-LSTM). We expect that Bi-LSTM and CNN would capture different information so their combination would help to enhance the overall ED performance. Let, there are k numbers of predefined event types $E_1, E_2, ..., E_k$. Now, given a sentence $S = w_1 w_2 w_3 ... w_n$, where n is the sentence length, for each word w_i in the sentence, we want to predict whether the current token is an valid event trigger. And if it is an event trigger then with which event type E_j the word w_i will be matched. So, the current word along with its sentential context constitute an event trigger candidate which will be input to our classification model. We have fixed the context window size in order to feed the trigger candidate to the CNN. If the window size is k then trigger candidate for word W would be represented as $[w_{-k}, w_{-k+1}, ..., w_0, ..., w_{k-1}, w_k]$ where current word is positioned in the middle position (i.e. w_0). Now these $(2k + 1)$ size trigger candidates are taken as input to the CNN and Bi-LSTM models. Before entering the CNNs and Bi-LSTMs, each word is converted into a real valud vector by looking up a embedding table with an intution that these real valued vectors will capture various symantic characteristics of the words. So the input to our model is a matrix X of size $(2k + 1) * |V|$ where $|V|$ is the dimension of the real valued vectors and $X = [x_{-k}, x_{-k+1}, ..., x_0, ..., x_{k-1}, x_k]$ where x_j is the vector representation of word w_j. Now the two dimensional representation of each word is feed to a convolution layer followed by a max-pooling layer. In convolution layer we have used a set of filters $F_1, F_2, ..F_p$ and each filter F_i has a window size m_i and can be represented as a matrix of size $m_i * |V|$. After employing max-pooling on the convolution layer output we get a hidden vector representation of size $|F_1| + |F_2| + ... + |F_p|$ where, $|F_i|$ represents the number of instances of filter type F_i. The hidden vector representation which we get from CNN model is named as F_CNN. Besides CNN we have also used Bi-directional Long Short Term Memory(Bi-LSTM) in this work. The matrix representation of each word W which is represented as $X = [x_{-k}, x_{-k+1}, ..., x_0, ..., x_{k-1}, x_k]$, is also feed to the Bi-LSTM model. The hidden states of Bi-LSTM are computed both in forward and backward ways at each time step. We denote the outputs of the forward and backward LSTM as $\overrightarrow{h_t}$ and $\overleftarrow{h'_t}$ respectively. At each time step t, hidden vector h_i is computed based on current input vector x_t and previous hidden vector h_{t-1}.

$$\overrightarrow{h_t} = \overrightarrow{LSTM}(\overrightarrow{h_{t-1}}, w_t) \tag{1}$$

$$\overleftarrow{h'_t} = \overleftarrow{LSTM}(\overleftarrow{h'_{t-1}}, w_t) \tag{2}$$

Then the output at time t is $h_t = [\overrightarrow{h_t}, \overleftarrow{h'_t}]$. Here we have taken the hidden representation of x_0 i.e. concatination of hidden vector representation of forward LSTM $\overrightarrow{h_0}$ And hidden vector representation of $\overleftarrow{h'_0}$ as the output of the Bi-LSTM. Formally, $F_{Bi-LSTM} = [\overrightarrow{h_0}, \overleftarrow{h'_0}]$. Now to combine the output of CNN and Bi-LSTM we have concatenated the representation vector F_{CNN} and $F_{Bi-LSTM}$ and feed the concatenated vector to a fully connected layer followed by a softmax layer to get the proper event type of the current word W. The gradients are calculated using back-propagation. We have also implemented regularization by dropout.

For Event-Argument Extraction we have applied the same model which is used for Event Trigger classification. Only the number of output classes is different in these two models. We have used BIO annotation scheme for argument identification as most of the arguments contain more than a single

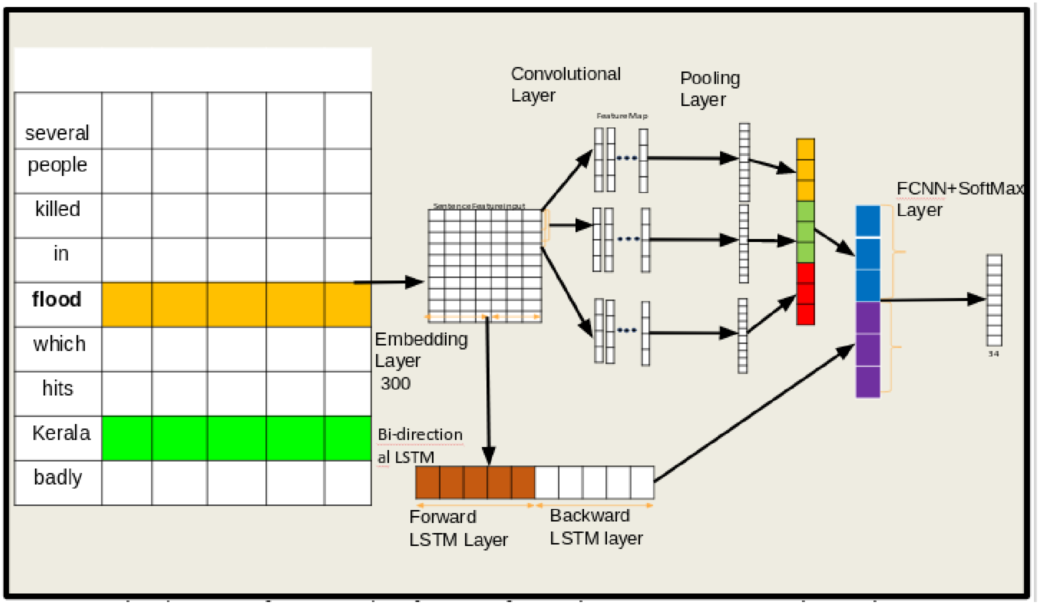

Figure 1: Deep neural architecture for Event Trigger classification

token.

Now after **Event-Trigger** and **Event-Arguments** have been identified, our next task is to link the event arguments (which are identified by the Event-Arument extractor) to their corresponding event trigger words so that we can identify the event participants for a specific event trigger. For **Event Trigger- Argument linking** we have taken a heuristic approach where we have calculated the relative distances between the argument and the trigger words present in a fixed window size of that argument. And then the trigger word with which the distance of that argument is less, the argument is mapped with that event trigger. The relative distance is measured as the number of word count between the trigger word and the argument phrase. And the fixed window i.e. the event trigger search boundary contains the sentence in which the argument exists along with the previous and next sentences.

3 Resources and Hyperparameters

We have used the same parameters for the neural network models used in trigger identification and argument extraction. In Convolution neural network we have used filter size of $2, 3, 4$ and for each size we have taken 200 filters to generate feature maps from the convolution operatons. The final output of the CNN is represented as F_{CNN} which has size 600. The number of hidden units used in Bi-LSTM is 200 which leads to the size of final outcome of the Bi-LSTM layer $F_{Bi-LSTM}$ is 400. Regarding embedding, we have used pre-trianed word embeddings of size 300 which we have taken from the fastText toolkit[1].

[1]https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md