Assignment – 4

Relation Extraction with Neural Networks

Anurag Dutt

SBUID: 113019209

Model

- ***Semantic Relation Classification:***

As mentioned in the Hendrickx et al. Paper, the aim of the assignment is to implement a Neural Relation Extraction Model, which extracts the relation classification from the annotated dataset of SemEval-2010 Task 8. The dataset contains nominal entity pairs and our task is to identify and classify the relation between the nominal entities. The task consists of implementing two functions:

- A Bi-directional GRU layer
- An attention layer as mentioned in the Zhou et al paper

The structure of the task and the model is as follows:-

1. Input layer: input sentence to this model;
2. Embedding layer: map each word into a low dimension vector
3. BiGRU layer: utilize BiGRU layer to get high level features from step (2)
4. Attention layer: produce a weight vector, and merge word-level features from each time step into a sentence-level feature vector, by multiplying the weight vector
5. Output layer: the sentence-level feature vector is finally used for relation classification

The main idea behind the BiGru layer is to introduce an adaptive gating mechanism which solves the vanishing gradient problem. We introduce bi-directionality to capture hidden temporal connections considering the sentence flow in the opposite order. Lastly, owing to the success of attentive neural networks, we introduce an attention layer which does the following transformations for a given matrix of output vectors as H = [h1, h2, ……..]:

- $M = \tanh(H)$
- $\alpha = \text{softmax}(wM)$
- $r = H\alpha$
- $h^* = \tanh(r)$

Lastly we add l2_regularization by constraining the L2 norms of the weight vectors by rescaling the weights after a gradient descent step.

$$L(\theta) = -\sum_i \log p_{t_i} + \frac{\lambda}{2}\|\theta\|^2$$

- ***<u>Model implementation:-</u>***

For implementation, we follow the Zhou et al paper. Firstly we initiate a bidirectional GRU layer, taking the input shape as a 2d matrix of size vocab_size, 2*input_dim (since we concatenate Word embeddings and POS tag IDs. In the call method we define a sequence mask for the paddings, which essentially eliminates the padding labels. We then concatenate the word embeddings and the POS tag IDs, and provide them as an input to our bidirectional GRU layer. We then supply the output of the BiGRU layer to the attention function, which essentially performs the following the four steps as mentioned in the paper:

- Takes the tanh of the BiGRU outputs
- Takes the dot product of the resultant with the omegas supplied
- Calculates the weights using a softmax function and takes the weighted sum.
- Returns the tanh of the weighted output.

The input size of the embeddings is 200, which is essentially concatenation of the 100 dim glove word embeddings and the 100 dim word tag IDs. All of the above implementation is done in the Lastly we implement a l2 regularization step in train_lib.py, where we use 10^(-5) as our lambda as is given in the paper. The regularization is applied on all the trainable variables. The result of the three experiements conducted are as follows:-

<u>Experiment</u>	**<u>Configuration</u>**	**<u>F1</u>**
Experiment 1	Word Embeddings Only	0.5163
Experiment 2	Word embeddings + Pos Embeddings	0.5232
Experiment 3	Word embeddings + dependency structure	0.6056

From our observation it is very clear that pos embeddings add very little weight to our classification task, however add the dependency structure increases the F1 score of our classification by a significant

margin. We can see this observation because the shortest dependency path combined with the BiGRU model picks up on the heterogeneous information between the two entity nominal pairs. However, just adding word embeddings or combining word embeddings and pos embeddings is not helpful, as much as it just regards the raw text as a sequence. This observation might occur due to two primary reasons:-

1. The shortest dependency paths retain most relevant information (to relation classification), while eliminating irrelevant words in the sentence.
2. The multichannel LSTM networks allow effective information integration from heterogeneous sources over the dependency paths.

Adding only POS tags might not make that big a difference because POS tags are essentially discrete and coarse-grained meaning that only a limited number of POS tags can be used. Hence it might be very difficult to capture the entire semantic meaning of the sentence using just POS tags, we might need to append the the dependency structure information to improve our results.

Advanced Model

For the purposes of out implementation, we try and implement the following CNN model as described in the paper Relation Classification via Convolutional Deep Neural Network by Zheng et al:-

- ***Architecture***

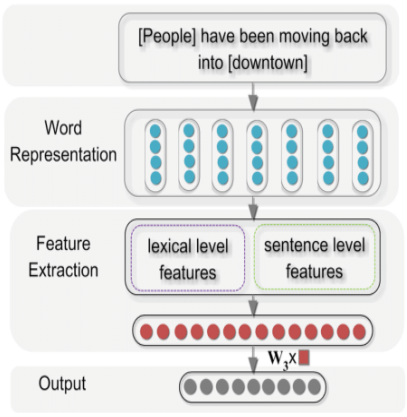

Figure 1: Architecture of the neural network used for relation classification.

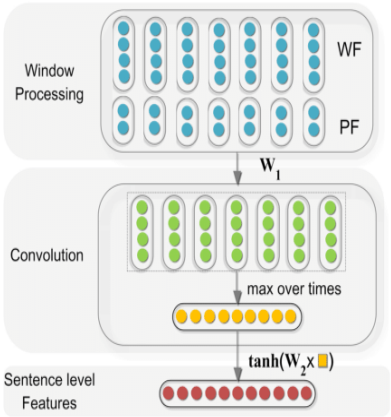

Figure 2: The framework used for extracting sentence level features.

The network takes an input sentence and discovers multiple levels of feature extraction, where higher levels represent more abstract aspects of the inputs. It primarily includes the following three components: Word Representation, Feature Extraction and Output. To identify the relations between pairs of nominals, we combine the following lexical and sentence level clues from diverse syntactic and semantic structures in a sentence:

- ***Word Features***

100 dim pretrained glove embeddings act as word embeddings

- ***Position Features***

Position Features are proposed for relation classification in the CNN model. In this implementation, the Position Features are a combination of the relative distances of the current word in the sentence to the given entities e1 and e2. For example, in the sentence S : [People] have been moving back into [downtown], where the entities are [People] and [downtown] the relative distances of "moving" in sentence S to "people" and "downtown" are 3 and -3, respectively. The relative distances also are mapped to a vector of dimension which is randomly initialized. Then, we obtain the distance vectors d1 and d2 with respect to the relative distances of the current word to w1 and w2. Thus we get two tensors for each sentence corresponding to the relative distances or position of each word in the sentences to each of the two entities. The position features are concatenated to the word features to get a 300 dim embedding vector for each word. W

Combining the Word Features and Position Features, the word is represented as [WF, PF1, PF2], which is subsequently fed into the convolution component of the algorithm.

- ***Implementation:***

To get the position, I had to change the structure of the code a bit. I needed to create a position matrix marking relative distance of each word in the sentence to the two entities. As such I needed to pass the text_tokens along with word IDs and POS IDs to my call method. I made changes to the function "generate_batch" where I add the actual text tokens to each batch size as an input to call method.

I identify the entities by the suffix and prefix, "<e11> and <e12>" for the first entity and "<e21> and <e22>". I calculate the relative position of each word in the sentence, once I have identified the entities. Once I have the distance embeddings, I concatenate the embeddings to the actual glove embeddings of the word and pass the embeddings to a Convolution 2D layer, followed by a GlobalMaxPool2D layer

and lastly apply a dropout of 0.4 and then a final softmax layer for classification into num classes which is provided as an user input. The hyperparameters of the conv2d layer is as follows:-

- num_filters = 64
- kernel_size = 3
- activation = “relu”
- input_shape = (None, None, 1)
- padding = “same”

I needed to pass an additional input in form of text tokens to identify the entities in the sentences.

- ***Justification:***

Relation classification can be considered a multi-class classification problem, which results in a different objective function. Relation classification can also be defined as assigning relation labels to pairs of words. It is thus necessary to specify which pairs of words to which we expect to assign relation labels. For that purpose, the position features are exploited to encode the relative distances to the target noun pairs. The relative position features provide a great representation of lexical features corresponding to the two entities. Although we can use feature based methods for accomplishing the task of relation classification, fKernel-based methods provide a natural alternative to exploit rich representations of the input classification clues, such as syntactic parse trees. Kernel-based methods allow the use of a large set of features without explicitly extracting the feature-based methods suffer from the problem of selecting a suitable feature set when converting the structured representation into feature vectors. The method described in paper uses a convolution kernel to tackle the aforementioned problem.

- ***Results:***

In our case we observe a small improvement, in the validation F1 value when I train the CNN model. In the BiGRU task the observed F1 for the best case model using word_embeddings+SDP is 0.6. The best case for the CNN model is close to 0.66, which is almost a 9% increase in performance than the existing BiGRU model. The three best case experiments are:-

1.) Training the CNN model for 8 epochs, with dropout 0.4

2.) Training the CNN model for 8 epochs, with dropout 0.2

3.) Substituting the Conv2D kernel with Conv1D filter with the same hyperparameters, with the only change being number of filters being increased to 128, dropout = 0.2

The observed results are as follows:-

Experiment	**Configuration**	**F1**
Experiment 1	CNN 2D – 8 epochs, dropout = 0.4	0.6583
Experiment 2	CNN 2D – 6 epochs, dropout = 0.2	0.6481
Experiment 3	CNN 1D – 5 epochs, dropout = 0.2	0.6029

The predict files are named correspondingly for each of the experiements. I would also be submitting my data.py file as file as I made changes to two functions namely ***index_instances*** and ***generate_batches,*** so that I could extract the actual text tokens. There was no way for me to calculate the position of entities, without knowing the actual text tokens. (Technically I could have assumed the entities are generally nouns and noun analogous terms and looked up the corresponding POS tag IDs in the global POS ID matrix, but this method relied on a very big assumption that entities have certain POS ID tags, I found it much more reassuring to actual extract and verify the corresponding information from the actual text tokens and hence the change to the two functions in data.py.

Shared Drive Link
https://drive.google.com/open?id=1oJp-TuR-MmDEGANGleF4RF1jUsGz4Un0

Wget link:-
https://docs.google.com/uc?export=download&id=1oJp-TuR-MmDEGANGleF4RF1jUsGz4Un0