NLP Assignment 4: Relation Extraction
CSE 538 Fall 2019

Anmol Shukla, SBU ID - 112551470

15^{th} December 2019

1 BiGRU + attention implementation

For my implementatation, I have followed the apporach as described by Zhou et al. The architecture for the model looks as follows -

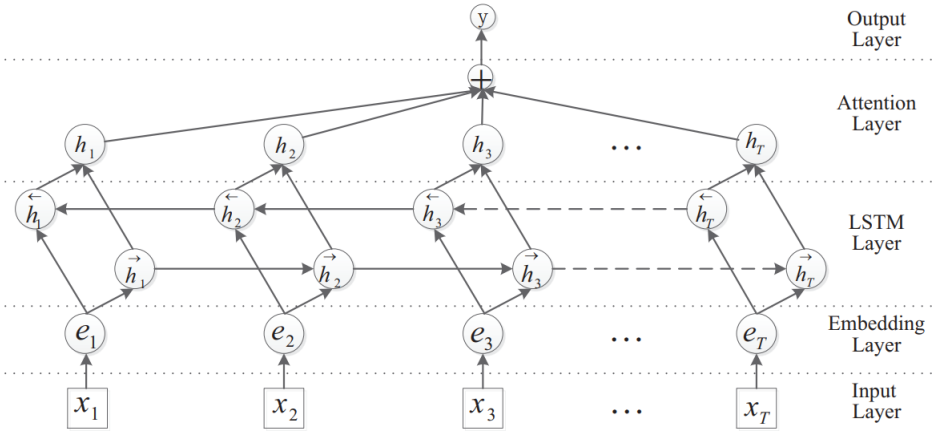

Figure 1: Architecture of baseline model i.e Bidirectional GRU with attention, reference Zhou et al

First, we will define the following layers in the __init__ method of our **MyBasicAttentiveBiGRU** model

- *_forward_layer*: The forward layer computes the forward pass on the given input sequence. It is defined using *layers.GRU* and contains 128 GRU units. Since we want to return the full output sequence from the forward layer to the backward layer, I have set *return_sequences = True*. Additionally, I had also tried using *he_normal* initializer for this layer but it gave me a lower F1 score.
- *_backward_layer*: This layer is declared similar to the forward layer, the only difference being that I have set the *go_backwards* flag as True so that the layer process the input sequence backwards and returns the reversed sequence.
- *_bidirectional_layer*: This is a keras.layers.Bidirectional layer which wraps the forward and backward layer together. The *merge_mode* is set to "concat" since we want to concatenate the outputs of forward and backward layer (although the paper sums up the outputs of forward layer and backward layer, TAs have mentioned to only concatenate them).

Then, I have implemented the attention function **attn()** as follows:

- First I have applied the tanh() function to the output of our BiGRU (equation 9 of the paper, $M = tanh(H)$).

- Then I have taken a dot product between the omegas and the output of the last step along the time-steps to compute α (equation 10 of the paper, $\alpha = softmax(w^T M)$)
- The next step is to compute a softmax of the previous output and then apply a tanh function on the softmax output equation 11 and 12, $r = H\alpha^T, h^* = tanh(r)$)

The **call()** function is implemented as follows:

- The first step is to concatenate the word embeddings and its corresponding POS embeddings. This is done by using tf.concat along the third axis i.e. axis = 2.
- Then we need to create a mask as our input sequence is padded with 0 (since not all sentences have equal no. of words). This is done by checking which words have a 0 word ID. This mask is then passed to the bidirectional GRU layer later on.
- Then we call the bidirectional GRU layer by passing the concatenated word and POS embeddings and the mask.
- The output of the bidirectional layer is given to the attention layer which calculates attention using attn() method.
- Lastly, we pass the attended outputs to a dense layer (decoder) which gives us the logits (size - batch_size x num_classes)

L2 Regularization loss: For calculating the L2 regularization term, I have first obtained a list of all the trainable parameters of my model using *model.trainable_variables* and then used tf.nn.l2_loss on every parameter to obtain the l2 norm. The sum of all the l2_loss multiplied by regularization parameter λ will be the overall regularization term.

2 Experiments

2.1 Default Configuration

With the default configuration i.e. using word emebeddings, POS emebeddings and dependency structure, I achieved an F1 score of *0.613* in 5 epochs. I tried out several things such as dropout, different initializers for recurrent layers before I achieved this F1 score. The score is in line with the TAs expectations.

Epoch	**Loss**	**F1 Score**
1	2.1520	0.3352
2	1.8389	0.4225
3	1.6695	0.5118
4	1.7026	0.5831
5	1.788	**0.6130**

2.2 Experiment 1: Only Word embeddings

Using only word embeddings as input features gave me a relatively low F1 score of *0.4612* when compared to the default configuration. This was expected, as we do not have the dependency structure and the POS tags which would help us in entity extraction and subsequent classification of relations between the entities.

Epoch	Loss	F1 Score
1	2.387	0.2816
2	1.9939	0.3605
3	1.7969	0.3885
4	1.8953	0.3922
5	1.9288	**0.4612**

2.3 Experiment 2: Only Word + POS embeddings

Using only Word and POS embeddings did even worse than using just the word embeddings. The 5 epoch validation F1 score was *0.3388*. I observed that the score fluctuated between epochs and the 5 epoch score was even lower than experiment 1. This shows that just using POS embeddings with word embeddings hurts rather than helping. One possible reason could be that the model pays more attention to the POS embeddings with lack of dependency structure and therefore ends up miss-classifying many sentences just by judging on the basis of word and POS embeddings. For example, the model might see that having a Noun-Noun in the sentences expresses the presence of certain relationship, which is not true. Presence of two specific noun words need not mean that there is a relationship between them. We might need the dependency structure too.

Epoch	Loss	F1 Score
1	2.9197	0.3426
2	2.5681	0.3656
3	2.2136	0.3298
4	2.1211	0.2806
5	2.0358	**0.3388**

2.4 Experiment 3: Only Word embeddings + Dep structure

Just by using Word embeddings and dependency structure, I got a similar F1 score to the default configuration. I think that dependency structure has a higher importance than the POS tags because dependency structure actually gives the dependency between different words in our sentences which can help us to identify relations. This proves that it is important to have the dependency between the words to correctly identify the relationship. Alternatively, we can say that presence of two words such as "Farmer" and "crops" does not express a Producer-produces relationship. The dependency structure could help us to identify if there actually is some relationship.

Epoch	Loss	F1 Score
1	1.8937	0.3851
2	1.5887	0.5108
3	1.5317	0.5768
4	1.6259	0.5759
5	1.7423	**0.5829**

3 Advanced Model: CNN based architecture

Before building the final advanced model, I tried out almost 20 different models. Some of the architectures that I tried out are as follows -

- Conv1D + GlobalMaxPooling1D + Dense + Decoder
- Conv1D + MaxPooling1D + LSTM/GRU + Decoder
- Conv1D + MaxPooling1D + BiGRU/BiLSTM + Attention

- Conv1D + MaxPooling1D + Conv1D + MaxPooling1D + Dense + Decoder

I also tried out dropout at many levels and also tried stacking multiple models and then combining their output for final prediction. In the end, I found stacking 3 CNN layers to be the best as detailed below.

Final Model : Initially, I tried building the advanced model using a single convolution layer followed by an LSTM/GRU. I noticed that varying the number of filters and size of the kernel had a slight impact on my F1 score. Thus, I took the idea of ensemble learning and stacking three convolution layers together which will have different number of filters. This way, all the three layers will be able to learn different representations of the input and the model as a whole, will be able to select the best representation by back-propagation mechanism. The layers of the model have been declared in the __init__ method of **MyAdvancedModel**. The layers are as follows -

- *_masking* layer: We need to mask out the words which are essentially paddings i.e. 0. We can do this by creating a mask similar to the basic model and passing that to the Keras layer.
- 3 *Conv1D* layers: I have created 3 Conv1D layers for the three CNN layers which will be stacked together. I have used tanh activation function as I found out that tanh led to faster convergence than relu. I have also kept the kernel_size as 3 which gave a better results than a size of 2.
- 3 *GlobalMaxPool1D* layers: After the Conv1D layers, we need to pool the outputs. I have used GlobalMaxPool1D instead of MaxPool1D as we need to pool across the time-steps i.e. words so that we have an output of the shape batch_size x #filters.
- *Concatenate* layer: We need to concatenate the outputs of all the pooling layers which can then be passed to the dense layers.
- 2 dense layers: I created two dense layers - (1) With 100 units for the concatenated outputs of CNN (2) With 19 units for the logits.
- *Dropout* layers: I used 2 dropout layers - (1) For input (2) For output (before the first dense layer). I found dropout to be really useful as it gave me a good F1 score for validation set due to its regularizing effect. But if dropout ¿ 0.5, I started getting NaN due to high dropout of units.

3.1 Architecture

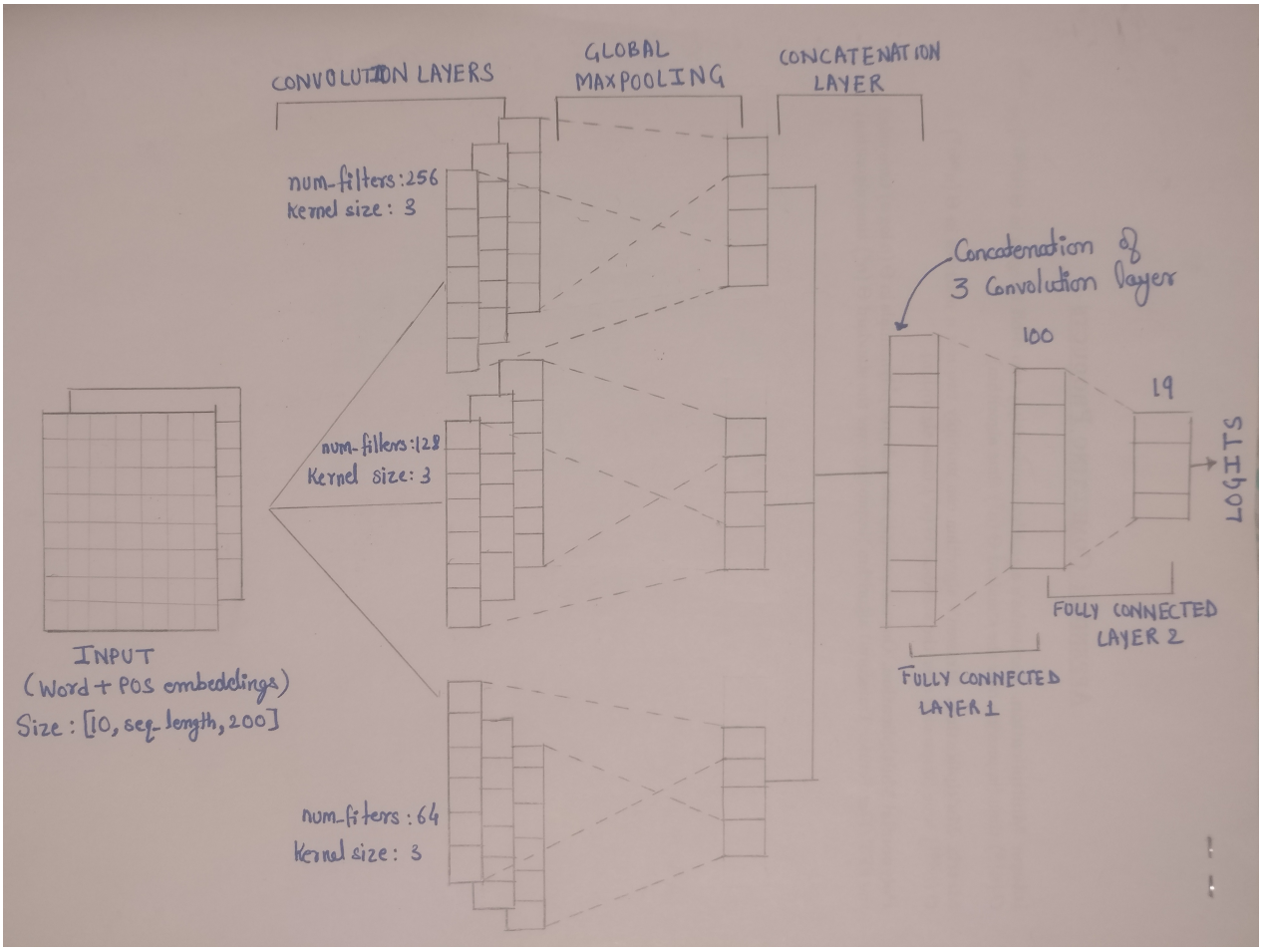

Figure 2: Final Model Architecture: 3 different CNN layers stacked together

I have also attached the model architecture image in the submission zip file, please take a look at the same for a better/larger picture.

Performance: I achieved an F1 score of *0.6507* on validation data after training for 17 epochs.

Justification: I evaluated different versions of CNN models as per Matt's suggestion on Piazza. As described earlier, I followed the approach of stacking the 3 CNN models together inspired by ensemble learning as I believed that each CNN model will learn a different representation of the input sentences and the model will figure out the best representation/features from there. The CNN filters will help us to detect patterns in our sentences such as whether two words appear together or not in a window. For example, if two related entities appear together in a region, the value of applying a filter to that region will be higher. We then apply a GlobalMaxPool which will allow us to keep information about whether or not the feature appeared in the sentence, but we will lose out the information about where exactly it appeared. Therefore, CNNs are good for extracting local and position-invariant features. However, there can be one problem with CNNs - just the mere occurrence of two entities does not mean that the sentence express a relation between them. This is where the dependency structure and POS tags help us especially the dependency structure. I also found out that CNNs took relatively less training time than BiGRU which is an advantage as it allowed me to iterate over different hyperparameters more quickly.