GRU Implementation :-

Model Implementation details -

Explanation of GRU Implementation - The implementation of this model was inspired by the paper mentioned in the assignment pdf [1]. The language modelling task that this model tries to solve is to classify the relations between two keywords in a given sentence. This was achieved by augmenting the textual data with the 100 dimensional GloVE word vectors. The implementation details are as follows -

Explanation of GRU -

1. Due to lack of resources, I tried to keep the model as simple as possible. Therefore, I just went with implementing only one Bi-Directional layer with a single GRU unit.
2. The GRU unit has a Sigmoid recurrent activation function.
3. While initiating the forward pass, as per the directions given in the assignment pdf, I built a baseline model by concatenating together the *'word_embed'* and *'pos_embed'* and also maintaining the information about the *'dep structure features'* loaded in *'data.py'* file.
4. Additionally, masking was also implemented into the code, where the GRU layer consumes the mask that contains only active states of the input.
5. In order to carry out the 3 experiments listed in the assignment pdf, I made appropriate changes to the code and noted down the observations. I made the following changes to carry out the three experiments -

```
# Normal mode - data = tf.concat([word_embed, pos_embed], axis=2)
# Only word embedding features ---- data = word_embed (with dep removed in data.py)
# Dropping word + pos features ---- data = tf.concat([word_embed, pos_embed], axis=2)
# (with dep removed in data.py)
# word embedding + dep features ---- data = word_embed (with dep not removed in data.py)
```

Attention Implementation -

1. I followed all the steps mentioned in the paper *Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification* [1]. Section 3.3.
2. I have also actively incorporated the modifications into the model mentioned by the TAs on the piazza posts from time to time.
3. The output of the attention layer was finally fed to the *'decoder'* layer aka a dense layer with softmax activations.

The model summary that I built looks like the following -

Model: "basic_attentive_bi_gru"

Layer (type)	Output Shape	Param #
dense (Dense)	multiple	4883
bidirectional (Bidirectional	multiple	253440

Total params: 678,979
Trainable params: 678,979
Non-trainable params: 0

Observations -

1. Basic Model -
    a. I found that the basic model which included the *'word_embed'*, *'pos_embed'* and the *'dep structure features'* performed the best among the 4 basic bi-directional models that I trained (1 basic model + 3 models from 3 experiments)
    b. The model had an **F1 score of 0.6068 after 5 epochs**.
    c. The predictions of this model are submitted as the final predictions of the basic model.
2. Experiment 1 -
    a. In this model, I removed the *'pos_embed'* and the *'dep structure features'* and trained the model on only the *'word_embed'*.
    b. Given the F1 score of this model, it appears that the 'word_embed' are indeed power enough to give moderate performance on their own.
    c. The model had an **F1 score of 0.4341 after 5 epochs**.
3. Experiment 2 -
    a. In this model, I removed the *'dep structure features'* and trained the model only on *'word_embed'* and *'pos_embed'.*
    b. As per my observations and experimentation, I found this model to be the worst performing models among the 3 experimental models.
    c. The model had an **F1 score of 0.4062 after 5 epochs**.
4. Experiment 3 -
    a. In this model, I removed the *'pos_embed'.* I trained the model on the *'word_embed'* and *'dep structure features'.*
    b. As per my observation and experimentations, I found this model to be the best performing model among the 3 experimental models.
    c. The model had an **F1 score of 0.5846 after 5 epochs**.
5. Conclusions -
    a. From the above observations, I can see that the pair *'word_embed'* and *'dep structure features'* is the most informative pair when asked to use only 2 of 3 data pieces.
    b. Similarly, I observed that *'word_embed'* and *'pos_embed'* is the least informative pair when used in combination.

Justification or motivation for an advanced model -

My implementation is influenced by the papers [2] and [3]. I have implemented a basic DNN using a single CNN layer coupled to the GlobaMaxPooling. The main reasons for this design choices are as follows -

1. CNNs perform a discrete convolution on the input matrix with a set of different filters. In my implementation, I mainly tried using 32 filters with a size of (2, 2) in 2 dimensions. I also tried different number of filters, but I found the 32 filters gave me the best validation F1 score overall when compared to the other models with different filter numbers.
2. For NLP tasks, the input matrix represents a sentence: Each column of the matrix stores the word embedding of the cor-responding word. By applying a filter with a width of, e.g., three columns, three neighboring words (tri-grams) are convolved. This follows with the MaxPooling layer which select the 'main' keywords out of these tri-grams.
3. Pooling is a common concept while processing images, but there the input sizes are fixed. So, a simple MaxPooling2D() layer can do the job. But in NLP tasks, sentences could be of variable lengths, meaning that the input size could vary from sample to sample. Given that padding may work fine in this scenario, I resorted to using a special function 'GlobaMaxPool2D' from TensorFlow, which is kind of able to handle the variable sized inputs.

4. Most of the times, the Convolutional layers are followed by flattened layers, which need a fixed size input, 'GlobaMaxPool2D' is crucial to facilitate this. Otherwise, one could only build a DNN with just the CNN layers with the last layer in the network containing the number of filters is equal to the number of predicted classes, followed by a pooling layer to attain the same effect.
5. The max-pooling extracts the maximum value for each filter and, which is the most informative n-gram from the following steps. Finally, the resulting values are concatenated and used for classifying the relation expressed in the sentence.
6. This paper [2] introduces a concept called *'Extended Middle Context'* where the authors hypothesize that the contexts are split into three disjoint regions based on the two relation arguments: the left con-text, the middle context and the right context.
7. Mostly the middle part of the sentence, contains the most crucial information that connects the either side semantics. The two contexts proposed by the authors are - (1) a com-bination of the left context, the left entity and the middle context; and (2) a combination of the middle context, the right entity and the right context.
8. These two contexts can be easily processed by the CNN and max pooling. CNN is also able to easily able to catch the meaning which are spread over the sentences. This is possible through the pooling layer.
9. I could also experimentally verify that the Advanced Model that I designed using CNN and Max Pooling could easily outperform the Basic attention based bi-directional model that I built earlier.
10. The F1 score for **5th epoch for this model was - 0.6618** against the **basic model which had an F1 score of 0.6068 at 5th epoch** in its best configuration.

Advanced model looks like the following -

Model: "my_advanced_model"

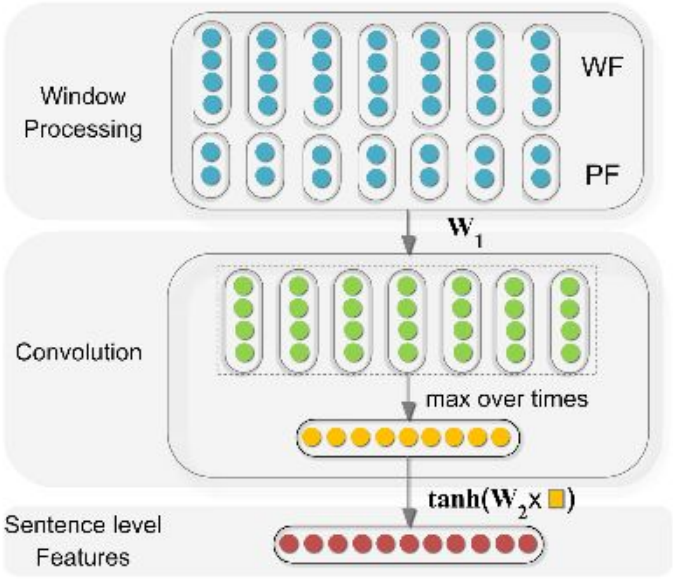
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	multiple	320
global_max_pooling2d (Global)	multiple	0
flatten (Flatten)	multiple	0
dropout (Dropout)	multiple	0
dense (Dense)	multiple	627

Total params: 421,347
Trainable params: 421,347
Non-trainable params: 0

None

Due to use of 'GlobalMaxPooling' the count of parameters for the layers after it and before the last Dense layer are dynamic and vary as per the input size. Therefore, are mentioned as 0 above.

Figurative illustration of advanced model -



A simple illustration of the working of CNN and pooling layers which enable sentence level feature extraction. [2]

Some basic experimental observations with the Advanced Model -

Epochs	Val F1 Score	Dropout	F1 Score
5	0.6618	0.1	0.6618
8	0.6910	0.4	0.6047
10	0.6315	0.5	0.5
15	0.6164		

I tried to tune the advanced model, by using simple techniques. Through which, I could find that the best epoch for training the model was 8 and the best value for hyperparameter dropout was 0.1. Output corresponding to model trained on this configuration is attached in the submission.

References -

[1] *Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification* (*https://www.aclweb.org/anthology/P16-2034.pdf*)
[2] *Combining Recurrent and Convolutional Neural Networks for Relation Classification* (*https://www.aclweb.org/anthology/C14-1220.pdf*)
[3] Relation Classification via Convolutional Deep Neural Network (https://www.aclweb.org/anthology/C14-1220.pdf)
[4] How to implement a basic CNN with tensorflow - (https://towardsdatascience.com/tensorflow-2-0-create-and-train-a-vanilla-cnn-on-google-colab-c7a0ac86d61b)