CS695 PROJECT (PHASE 2)
EMPHASIS SELECTION BY PRE-TRAINED ERNIE LANGUAGE MODEL

By

PHANI KRISHNA KARRA(G01188931), SRIHASA VEJENDLA(G01167831), MITHILAESH JAYAKUMAR(G01206238)

Introduction: Our project "Emphasis selection using pre-trained Ernie model" mainly focuses on predicting the words that need to be emphasized in each sentence. It returns the probabilities for each word in the sentence and we consider on emphasizing the top 4 words in a sentence that have high probability values. Emphasizing the words in a text is very much helpful in visual media as visual media serves as a platform containing a lot of advertisements, flyers, posts, and motivational quotes. We believe that by emphasizing specific words a huge impact is created on the viewers. We made use of the pre-trained models like Ernie that are already pre-trained and available to design a model that gives us the probability for word emphasis.

Requirements: Our model requires at least 16GB ram and a working GPU to ensure a smooth execution. If not, our model might crash in the middle and might not be giving the results which, we obtained.

Duration for execution: We used a variation of ERNIE models such as large (1024 layers) and normal (768 layers) by using the Hugging face transformer library. It took around 15 to 30 mins for a single training epoch and around 2 to 3 mins for every validation epoch based on the Ernie large or normal model. It might take around 3 to 4 hours for the complete training of the model that we created. This model was trained for 10 epochs with early stopping for a patience of 4.

Dataset: The dataset has 2742 training instances, 392 dev instances and 743 test instances in which each instance is a sentence with varying number or words in it. The train data has words and their emphasis labels (B – Must emphasize, I – Consider for emphasize, O – Not consider for emphasize), POS tags of the words, the probability of each word (which is calculated by formula (B+I)/(B+I+O)). Instead of predicting the class labels B, I, O, we directly regress the probabilities using our model and compare it with the given probabilities of the words. We make the model learn by this comparison to make it predict more accurately. We keep the track of the POS tags as we believe that they might be helpful in our future implementations for improving the model.

Pre-processing: We took the words, POS tags and probabilities of the words and removed any other columns that we felt are not useful for our implementation and then separated each of them into a list. We then tokenized the words by using Ernie tokenizer which even does sub tokenization. Sub tokens means, dividing a word into multiple words to get better emphasis. For example, the word 'Plantgang' is divided into plant and '##gang'. We then encode the tokens using the encoder which converts the string values of tokens into integer values. Both the tokenizer and encoder are of the ERNIE transformer model. We take the probabilities of the words in a list and re shape it according to the size of the generated encoded values. We group the train data into batches which decreased the execution time of the model by a huge margin. Before batching, a single epoch on train data took 90 minutes and after batching it took 20 minutes for a single epoch. We take the largest sentence in a batch and pad the remaining sentences accordingly. We also implemented early stopping which stops training the model when minimum validation loss is obtained.

Data Augmentation: We encountered a potential problem of overfitting as our training data was not enough and the model did not initially perform well on the validation data and test data. So, we added few data augmentation techniques. We increased the size of our training data from 2742 to 5500 and are planned to increase further to 8000 for the next phase. We see a slight delay in overfitting because of this and proven to be useful. Some of the data augmentation techniques used are.

1. Randomly removed a word
2. Randomly capitalizing the first word
3. Randomly reversing a sentence

We then shuffle the data and again shuffle the data for every batch so that the model does not overfit and generalizes well.

Lexical Features: In addition to the output features generated by Ernie, we tried to add additional features which we thought to be useful. We added the features vectors with values [0,1] based on conditions such as if the word starts with a hashtag, if the first letter is capital, if the whole is capital, if the word can be sub tokenized and if the word is an connector (article, preposition, etc.,) or not. We believed that adding additional emphasis based on the above features increased the probabilities predicted by the model. The feature vector was given values based on below conditions.

1. Feature vector '0' if a word is not capital, '1' if it is capital.
2. Feature vector '0' if a word does not start with '#', '1' if it starts with a '#'.
3. Feature vector '0' if the first letter of the word is not capital, '1' if the first letter of word is capital.
4. Feature vector '1' if the word can be sub tokenized, '0' if not.
5. Feature vector '0' if the word is a connector, '1' if the word is not.

Implementation: After the initial steps like data augmentation, pre-processing, feature vector creation, tokenizing and encoding, the data is sent to the model in batches which are padded. The model class passes the data to Ernie model for contextual embedding and then the outputs generated by the Ernie are passed to a linear layer. We skip the hidden layers output and only consider the embedded output layer. We take the values of the output layer and pass it to linear layer which outputs a single layer from 1029 layers (including manually generated feature vectors). Then the outputs of linear layer are passed to a sigmoid function which converts the given outputs between 0 and 1. These outputs are our required probabilities which are then passed to the MSE loss function. This function computes the loss by comparing with the actual ground truth probabilities. The loss is calculated for every batch during every epoch and the model gradients are adjusted using a Adamax optimizer with a learning rate of 0.0001(Ernie large, Ernie normal) and 0.001(Ernie normal). This is repeated for the whole data for 10 epochs.

After the training of the model is done, we save the model, and the test data is then given to the model. We then obtain the test probabilities in accordance with the data which we trained on our model.

Loss: We used MSE loss as a metric for evaluating our model which finds the difference between the ground truth probabilities and the probabilities which our model gave.

What we did: We've done initial pre-processing, converted the words into tokens, encoded them, converted to tensors and then passed to the pre trained ERNIE model along with a linear layer and a sigmoid function

to get the outputs. We passed the train data to the model and trained it, then passed the test data sentences and obtained the test probabilities for each word. We also computed the top 4 words to be emphasized using a pairwise ranking algorithm.

What we are to implement further: We are planning to add features such as parts of speech of the word to make the model predict emphasis even more accurate. We are also planning to mix up the train data and dev data and implement K fold cross validation so that the data would be random for train and validation. By doing so the model would be able to learn the sentences better and we get better probabilities. This also ensures that there will not be any overfitting. We need to figure out more about the match metric and accuracy metric that is used as an evaluation for our model i.e. using the top m words in the match for emphasis and implementing it. We also want to tune the hyper parameters (learning rate) by changing them and different optimizers to observe how the model performance changes. We are also planning to compare our model's performance with various other models such as XML ROBERTa, BERT and ELMO embeddings instead of ERNIE and we will document which model is better relatively. Our current model was not tested by using many hyperparameters which we will be doing in the next assignment.

Challenges faced: Our main challenge was to pre-process the data according to the model and pass the data to the model. After passing the data to the model and getting the probabilities, we took some time to join the probabilities of the sub words to produce the data identical to the train data size for loss computation. Execution platform was another major challenge, and the code was crashing frequently during training. Even though our dataset has only around 3000 instances, our pre trained model took a lot of time to train especially Ernie large. Figuring out how to convert the outputs of the pre trained model by passing them to the linear layer to generate probabilities was another significant challenge. Another challenge is flattened probability values generated by the model. For the next phase we wanted to add normalization methods to make the probability values even more close such that they add up to a sum of 1.

We had our major challenge when the validation loss was not changing for all our epochs. We spent most of the time in figuring out this. We checked the code thoroughly and tried various hyperparameters, but we did not achieve any successful results. Finally, we figured out that the problem was the statement model.eval() which was given in an inappropriate place because of which the model stopped learning and did not make any improvement on the validation data though it was trained for many epochs. We rectified it and now we are getting a variation in the validation loss.

Individual contributions for the project are mentioned in the code as comments.