CS 695 FINAL PROJECT
EMPHASIS SELECTION ON WRITTEN TEXT IN VISUAL MEDIA

Phani Krishna Karra(G01188931)
Mithilaesh Jayakumar(G01206238)
Srihasa Vejendla(G01167831)

1. INTRODUCTION

Emphasis selection on written text in visual media is a task of predicting emphasis probabilities for the words in a sentence helping the user to choose which words to emphasize in order to have a greater impact on the people who are viewing the text. These days there are a lot of one liner sentences in use which have a lot of meaning such as advertisements, flexies, quotes in social media and even big companies have their captions written under their name in order to register themselves in the minds of the common people. Hence, we need to select the right set of words with correct features to be emphasized properly.

We develop a neural network model which takes the input sentence from the user along with the features which are obtained for each word in the sentence based on the word-case, grammar and hashtags. It gives us the emphasis probabilities for each word in the sentence as output. The top 'k' words are chosen for emphasis to give a better understanding for the user reading.

We tried various pretrained models like ERNIE 2.0 Base, ERNIE 2.0 Large, BERT Base Cased and BERT Base Uncased. Out of these pre trained models ERNIE 2.0 Large gave good performance in this task. We also tried an ensemble of BERT Base Uncased and ERNIE Base. But still ERNIE 2.0 Large outpowers BERT and our ensemble model in this task.

2. MOTIVATION

If we present the title of the project as "emphasis selection OF written TEXT IN visual media", it would not get any attention from the users looking at it because all the keywords that describe the task are not highlighted but the other words are highlighted. We thought that having a good model that can predict the correct emphasis for the sentences would be really helpful for generating impactful sentences for attracting the viewers.

3. DATASET

Our dataset has 3876 sentences on a whole out of which 1195 are from Adobe spark dataset and 2681 are from the Wisdom Quotes dataset. It has 44,976 words including both the datasets and 4886 unique words in it.

These datasets are divided into 2742 training, 392 dev and 743 test instances. The train data and the dev data have words, annotation given by 9 annotators with labels (B - Should Emphasize, I - Consider Emphasize, O - No Emphasize), POS tags of the words and the probability of each word (which is given by expression (B+I)/(B+I+O)). These B,I,O labels are given by showing the sentences to 9 separate annotators, each giving a label among the three possible labels based on their perspective. The probability is then calculated as shown above and those probabilities are used to train the models in our project. We did not consider the labels B,I,O because we already have the probabilities computed and won't be needing the labels in our working. There is also a column that shows from which dataset the sentence was taken(from Adobe spark or Wisdom quotes) which we ignore as it is not useful.

4. PREPROCESSING

As mentioned above, we removed the fields that we won't be needing for the project. We separated the sentences, POS tags and ground truth probabilities individually. Then we tokenized each sentence in which each word is tokenized and converted into tokens. The words which are the combination of two or more words (which are mostly adjectives, verbs and adverbs) are converted into sub tokens which form an important part of our project baseline implementation.

For example, the word 'beautiful' will get sub tokenized into three words 'beau', '##ti','##ful'. Sub tokenizing in this way helps the model for a better understanding and giving higher emphasis for these kinds of words. We use the tokenizer which is already available from HuggingFace library for each of the pretrained models that we want to test. The tokenized words are then encoded using the model specific encoder, converted to tensors, attention mask is obtained for each word and sent to the pretrained model.

We send the data to the model in batches of 32 sentences per batch and 64 sentences per batch for the Large(1024 layers) model and batches of 300 and 400 for small(768 layers). The training time of the sentences has been decreased by a huge margin due to batching. (A sample of 100 sentences took 150 seconds for a single epoch without batching and the same sample took around 40 seconds with batching). Even the whole data took around 20-25 minutes for training a single epoch. We also padded the sentences while sending the data in batches. Instead of padding the whole data to the length of the largest sentence, we padded the bits to the largest sentence in each batch which also proved to be efficient as it reduced the training time. We also implemented early stopping which is useful to store the best model and produce it instead of training the model for all the epochs thereby ensuring we got the best model.

5. ERNIE 2.0

There are two versions of ERNIE model, ERNIE Large and ERNIE Basel. The number of layers for the larger model is 1024 and for the smaller model is 768. ERNIE 2.0 is an improvement to the ERNIE 1.0 model with few improvements in the knowledge masking strategies which helps this model to achieve the state of art performance for this task. It is both entity level and phrase level language model. This model constructs three tasks namely word aware, structure aware and semantic aware tasks. By fine tuning this model, it adopts the kind of data that we are using for our task.

6. DATA AUGMENTATION

We used data augmentation in order to increase the size of the dataset to avoid potential overfitting problems as the size of our dataset is small. As we have very few instances of data, the model may not learn all the cases and conditions well. Hence, we tried the below data augmentation techniques and increased the size of the dataset to 5000 which was around 2800 before data augmentation. The training time increased after we used data augmentation since the size of the data is increased, but we also delayed the overfitting problem allowing the better performance of the model.

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

Performance of the model before and after the implementation of the data augmentation techniques can be seen at the results section.

7. FEATURE ENGINEERING

In addition to the vector embeddings that are already provided by our pretrained model, we manually tried to add a few features which we believed to be very much useful for our prediction using the final network. Feature Engineering means passing the information in the form of the feature vectors along with the vector embeddings given by pretrained models to our final network. These features help the model to decide on which words the model has to put additional emphasis.

We tried adding feature vectors [0 or 1] based on words starting with a capital letter or not, words that are divided into subwords(which means they have '##' in the word) or not, if the whole word is capital or not, if the word is a connection word (which means the word is a preposition or a conjunction or adverb) or not and if the word is a hashtag word or not. We also added parts of speech tags as a feature where we consider words tagged as verbs, adjectives, and Nouns. The addition of these features increased the accuracy of emphasis probabilities of the words predicted by the model.

1. Feature vector '1' if a word starts with capital, '0' if it is not.
2. Feature vector '0' if a word doesn't start with '#', '1' if it starts with a '#'.
3. Feature vector '1' if the word is fully capital, '0' if it is not.
4. Feature vector '1' if the word can be sub tokenized, '0' if it is not.
5. Feature vector '0' if the word is not a connection word, '1' if not.
6. Feature vector '0' if the POS tag is a pronoun, preposition, conjunction, det and '1' if the POS tag is a verb, noun or adjective.

8. IMPLEMENTATION

In our test data we only have the sentences. We do not have the POS tags and ground truth probabilities making it impossible to use the test data. So, we split and use the train data for training and validation purposes. We use development data for testing our model. After obtaining the sentences, POS tags and ground truth probabilities from our dataset, we use these for our model training and validation purposes. We send the train sentences one by one to the model. The model performs tokenization and encoding. The output is in the form of tensors for each word in the sentence which is then passed along with the attention mask to get the vector embeddings. Then the output is sent to a linear layer along with the feature vectors concatenated to it. A sigmoid function is applied on the linear layer output which gives us the probabilities for each word in the sentence. We run the model for 10 epochs where we believe that it would be enough for the model to learn all the parameters of the data we gave.

K- Fold cross validation is another method which is used to avoid overfitting and to increase the accuracy of emphasis probabilities. 8-fold cross validation has been used which is suggested in the state of the art work. Using cross validation, the data has been randomly divided into 8 folds where for each epoch, the data runs through 8 folds. In each fold, a single part of the data is considered as validation and remaining as train data. This data partition is changed in every fold so that the model learns from different parts of train data and validates on different parts of validation data. In this way, overfitting can also be avoided.

9.LOSS

In each epoch, the loss is computed for each fold of the data and we store the loss for all the folds. The average of the loss is computed for all the folds and is considered as the loss for every epoch. In this similar way we compute the loss for the 10 epochs. The same procedure is followed for the computation of validation loss. The metric which we used for the loss function is MSE loss which is used to compute loss between the ground truth values and the predicted probabilities which is given by running the model. The validation loss is considered for evaluating the model and we keep track of it using early stopping. If the validation loss is increasing, we do not consider the model and use the previous best model. We set the early stopping with a patience of '4'. The model stops its run when the validation loss is increasing continuously for 4 times. In this way, we use the MSE loss and early stopping to find the best model that gives us the best predictions.

10. PAIRWISE LOSS

When regressing the probabilities for words that are sub tokenized, we need to combine the probabilities of each sub word to find the predicted probability of the entire word. If we just average the predicted probabilities of each sub word then we ignore the relative scores between the subwords which might hurt the performance of the model. Hence, in order to overcome this we implemented the pairwise loss metric. This pairwise loss metric computes the loss between the subwords in the sentence and then computes a probability for the entire word.

11. EVALUATION

We used 'MAE' as our evaluation metric and saw reasonably good scores for our model. 'MAE' means 'Mean Absolute Error Loss', which tells us how much the model is deviating from the ground truth labels. We use the predicted probabilities that we obtained by giving the test data to the model and the ground truth probabilities to compute the loss on the test data for our model. Our model loss is ranging from 0.32 to 0.23 for different pretrained models that we used which is discussed below.

We also used a "match" metric for evaluating our model accuracy based on the top 'm' words. The top 'm' ground truth test probabilities which we have are compared to the top 'm' predicted test probabilities for each sentence which we obtained by giving to the model. We count the number of words that are matched in both the sentences and divide it with the value of m in top 'm' words to get the score for the sentence (rounded to 1). This step is repeated for all the sentences in our test data. We sum up the values obtained for all the sentences and divide them with the length of the dataset to get the 'match' score. (We used the top 5 words for our project).

Our match scores are ranging from 0.2-0.55 depending on the hyper parameters we gave and the model used. Our state of the art paper has reported the maximum score of 0.6 when they used words with '#' for emphasis and we reached a max score of 0.51 with similar configuration settings.

12. ANALYSIS

While trying different models and various hyper parameters for analysis, we observed a few things that influence the model in getting better output probabilities. The following are the important observations.

1. Using K-fold cross validation increases the run time of the model as it is divided into k folds and the number of sentences in the dataset also increases by 'k' times. K-fold cross-validation decreased the validation loss and increased the match score by a small margin. It also helped to delay and avoid overfitting of the model.
2. We used at most 5 fold cross validation for our data. We could observe the variation in the validation loss and the match metric when used 3 folds and 5 folds and thought of doing the 8 fold cross validation for the better scores but our systems were not supporting it and taking a lot of time for training. Hopefully we could improve our match metric if 8 fold cross validation is used since the model would be trained better and it learns well.
3. Another key observation is that we found a lot of similarities between BERT Base and ERNIE Base in their performance. A BERT Base model can also yield a similar

performance if the same configuration is replicated.

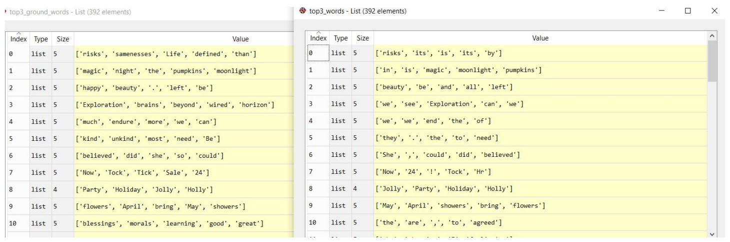
4. Using a dropout layer of 0.2 or 0.3 helped us to get well formulated probabilities instead of flattened probabilities. But giving 0.5 dropout did not work well for us as most of the words got 0.5 emphasis probability.
5. Using features were the most important part as mentioned in the research paper and we tried adding feature vectors based on the above mentioned conditions in addition to the vector embeddings generated by the model. We achieved our best results when we used feature vectors.
6. Using data augmentation helped our model from avoiding overfitting as we can see poor scores when we did not use data augmentation in our model.
7. In our test data we only have the sentences. We do not have the POS tags and ground truth probabilities making it impossible to use the test data. Hence we used dev data as test data for evaluation which decreased the size of the training data. In the state of art paper they combined both train and dev data and augmented it for training and validation purposes which might be one of the reasons for their good performance of the model.
8. We shuffled the data for every epoch, fold and batch which also helped to avoid overfitting to some extent.

12. OUTPUTS

A. Comparison of Various Models and Parameters

Model	K-fold cross validation	Optimizer	Data Augmentation	Feature Engineering	MSE loss	Match metric
ERNIE (LARGE)	NO	Adamax (0.0001)	YES	YES (NO POS tag)	0.31	0.40
ERNIE (BASE)	YES (3-Folds)	Adamax (0.0001)	NO	YES (NO POS tag)	0.27	0.35
ERNIE (BASE)	YES (3-Folds)	Adamax (0.0001)	YES	NO (NO POS tag)	0.35	0.22
ERNIE (BASE)	YES (5-Folds)	Adamax (0.0001)	YES	YES (NO POS tag)	0.28	0.42
BERT (BASE CASED)	YES (5-Folds)	Adamax (0.0001)	YES	YES	0.27	0.44
ERNIE (LARGE)	YES (3-Folds)	Adamax (0.0001)	YES	YES	0.26	0.47
ERNIE (LARGE)	YES (5-Folds)	Adamax (0.0001)	YES	YES	0.25	0.51
BERT (BASE UNCASE)	YES (5-Folds)	Adamax (0.0001)	YES	YES	0.26	0.46
ERNIE (BASE) + BERT (BASE UNCASE)	YES (3-Folds)	Adamax (0.0001)	YES	YES	0.25	0.47

B. Top-5 words ground truth and predicted



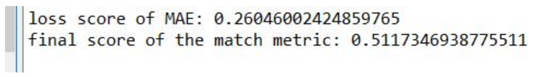
top3_ground_words - List (392 elements)

Index	Type	Size	Value
0	list	5	['risks', 'samenesses', 'Life', 'defined', 'than']
1	list	5	['magic', 'night', 'the', 'pumpkins', 'moonlight']
2	list	5	['happy', 'beauty', '.', 'left', 'be']
3	list	5	['Exploration', 'brains', 'beyond', 'wired', 'horizon']
4	list	5	['much', 'endure', 'more', 'we', 'can']
5	list	5	['kind', 'unkind', 'most', 'need', 'Be']
6	list	5	['believed', 'did', 'she', 'so', 'could']
7	list	5	['Now', 'Tock', 'Tick', 'Sale', '24']
8	list	4	['Party', 'Holiday', 'Jolly', 'Holly']
9	list	5	['flowers', 'April', 'bring', 'May', 'showers']
10	list	5	['blessings', 'morals', 'learning', 'good', 'great']

top3_words - List (392 elements)

Index	Type	Size	Value
0	list	5	['risks', 'its', 'is', 'its', 'by']
1	list	5	['in', 'is', 'magic', 'moonlight', 'pumpkins']
2	list	5	['beauty', 'be', 'and', 'all', 'left']
3	list	5	['we', 'see', 'Exploration', 'can', 'we']
4	list	5	['we', 'we', 'end', 'the', 'of']
5	list	5	['they', '.', 'the', 'to', 'need']
6	list	5	['She', ',', 'could', 'did', 'believed']
7	list	5	['Now', '24', '!', 'Tock', 'Hr']
8	list	4	['Jolly', 'Party', 'Holiday', 'Holly']
9	list	5	['May', 'April', 'showers', 'bring', 'flowers']
10	list	5	['the', 'are', ',', 'to', 'agreed']

C. MAE loss score and Final score of the model(Match metric)



```
loss score of MAE: 0.26046002424859765
final score of the match metric: 0.5117346938775511
```

13. CONCLUSION

By the end of this project, we were able to produce a model that takes in the sentences as the input and generates the emphasis probability as the output for each word in the sentence. Based on the predicted values we were able to get the top ‘5’ words in the sentence that are having the maximum emphasis probability.

Using 8 fold cross validation helped to achieve better results on the model. BERT and ERNIE had a lot of similarities and performance was also nearly the same. Data augmentation and feature engineering were the most important parts of the implementation.

As a further enhancement, we can try to scale the model’s performance for emphasis selection on texts in languages other than English.