# CSE 576 - Topics in NLP

# Final Project Report

# Logical Reasoning

Using Large Language Models with Tweaked Attention and Data Augmented from GPT-3

| | | | | |
|---|---|---|---|---|
| **Dr. Chitta Baral (Mentor)** | **Shivam Mathur 1222394875** | **Saurav Anchlia 1225472898** | **Sakthi Ganesh Mahalingam 1225347240** | **Md Nayem Uddin 1226315027** |

**Group Name: NSCube**

## 1. Introduction

Large language models are trained on vast amounts of data and are able to capture the statistical properties of language. This enables them to better understand the meaning of words and the relationships between words. Large language models have been shown to be effective at a number of tasks, including question answering, machine translation, and natural language understanding. They have also been applied to improve the accuracy of logical reasoning. Logical reasoning is a central component of many AI applications. Large language models can also help improve the accuracy of logical reasoning by providing a more comprehensive understanding of the context in which inferences are being made. They can also help to identify when a word is being used in a different way from how it is typically used. This can be important for understanding the implications of a statement.

We reduce logical reasoning problems to a natural language inference task *(Bowman et al., 2015)*. We generate datasets in the format of the premise, hypothesis, and labels for 42 types of logic. Utilize GPT3 to augment data and generate a train dataset. After establishing a baseline using a large language model, we tweak the attention mechanism to look for better results. We explore explainability tools like LIME to identify tokens that have an impact on prediction and utilize them to improvise the model performance.

## 2. Methods

**2.1 BERT-base-uncased**: BERT stands for Bidirectional Encoder Representations from Transformers *(Devlin et al., 2019)*. It is a transformer-based model that only uses the encoder part of the transformer architecture. We provide a more detailed explanation of BERT in Appendix A.1.

**2.2 RoBERTa-base-uncased:** RoBERTa is a Robustly Optimized BERT pretraining Approach *(Liu et al., 2019)*. This model is the same architecture as BERT. What is different is some of the pretraining strategies. Due to these strategies, it generally performs a tad better than BERT on popular benchmarks. We provide a more detailed explanation of RoBERTa in Appendix A.2.

## 3. Experiments

**3.1 Using Word Importance along with Attention Mask:** The attention mask in Transformer based models either has a value of 1 when attention needs to be applied to the token or 0 when the token is padding. This form of Hard-Attention works well in most cases, but considering the difficulty for language models to understand logical reasoning samples, we instead try a form of Explainable Attention Mask mechanism.

In our approach, we obtain word importance for sample predictions using a well-performing model using explainability libraries using LIME (Local Interpretable Model-Agnostic Explanations). We show how the LIME tool works in Figure 2 Appendix A3. These word importance values are then normalized between 0 and 1.

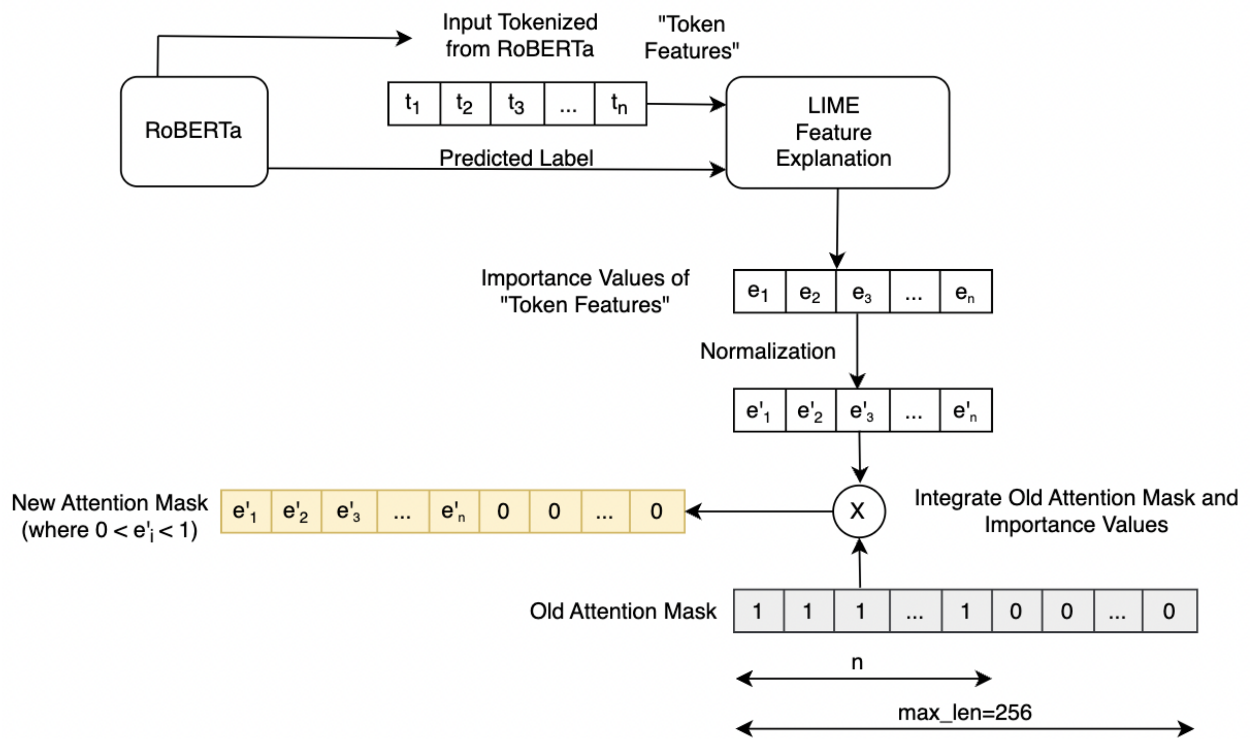

RoBERTa
Input Tokenized from RoBERTa
"Token Features"
t1 t2 t3 ... tn
Predicted Label
LIME Feature Explanation
Importance Values of "Token Features"
e1 e2 e3 ... en
Normalization
e'1 e'2 e'3 ... e'n
New Attention Mask (where 0 < e'i < 1)
e'1 e'2 e'3 ... e'n 0 0 ... 0
X
Integrate Old Attention Mask and Importance Values
Old Attention Mask
1 1 1 ... 1 0 0 ... 0
n
max_len=256

*Figure 1: Shows how the word importance is combined with the usual attention mask. The LIME tool outputs importance values for each token in the input text, which we normalize and combine with the usual attention mask. The new attention mask developed is of the same dimension as the old attention mask (max_len=256) but with soft values that are aimed at capturing explainability.*

Proposed Attention Mask = Attention Mask * Normalized Word Importance

(Refer Appendix A3, Figure 3, and Figure 4 for a comparison between old and new attention mask values)

Furthermore, the normalized word importance for the correctly predicted training and validation datasets is fed directly as the attention mask, whereas the word importance is inverted (1 - normalized word importance) for the samples predicted wrong.

We hypothesize that the baseline model (RoBERTa) would converge faster and would allow the model to understand important parts of the sentence better in order to make the correct predictions.

**3.2 RoBERTa-24 (Our modified Architecture):** The RoBERTa-base has 12 encoder and attention heads. We modify this architecture by adding another set of 12 encoders after the last encoder. The input given to the model is in the form of *premise +[SEP]+ hypothesis*

**3.3 RoBERTa- with Injected Hypothesis Input:** The same architecture as that of RoBERTa-24. The input here is given in the following format: *premise_1+ hypothesis + premise_2 + hypothesis + premise_3 + hypothesis +...+ premise_n + [SEP] + hypothesis*. Where *premise_1*, *premise_2*, .. *premise_n* are the various sentences in the premise, *n* is the total number of sentences in the premise.

**3.4. BERT Output Layer Experiment:** Bert has hidden states in all 12 layers; these are the outputs of each layer. While doing any classification task by default, we take the last output layer (a vector of 768 elements) and make a fully connected layer with the number of features. We tweaked the BERT output layer in two different experiments, where we took the last two output layers and the last four output layers for the classification task. So, concatenating the last layers produces 1536 elements, and the last four layers produce 3072 elements for the fully connected layer. This way, we can preserve more hidden layers of information while doing the classification, and it improves the baseline result.

**Our Understanding of Attention**

Attention [3] in an attention head is a function of three projections of the input sentence encoding. These 3 projections of the input are matrices - K, Q, and V. Q and K are of dimension - input length x dk. Hence, they sure may be different projections, but they are of the same input and, further, are projections onto the same dimension.

We analyze the term $Q_i K_i^T$. This results in matrix of dimension $d_{input} * d_{input}$. This matrix is basically measuring (or encoding) the relationships between the tokens of the input.

This is exactly where we attempt to modify or "tweak" our attention mechanism. We try to make modify this $Q_i K_i^T$ matrix and carry out experiments. We take inspiration from the methodology used in K-BERT wherein a Visibility matrix is added to $Q_i K_i^T$ term to tweak the self-attention. We try relatively simpler approaches to the same. Inspiration for our attention-tweaking thoughts was sought from *(Liu et al., 2020)*

**3.4 RoBERTa-x2 attention and RoBERTa-x100 attention (Tweaked attention approaches):**

We enhance the effect of relationship encoding amongst the input tokens by increasing every element of the $Q_i K_i^T$ matrix by a factor of S. This will further be normalized by $d_k$ and softmaxed to compute the final value of attention in a head. For the purpose of ease of understanding, we call this RoBERTa-x2. Note that this change is applied to all 12 attention heads. S takes the value of 2 and 100 in RoBERTa-x2 attention and RoBERTa-x100 attention, respectively.

## 4. Results and Analysis

| Experiment | Overall (All Labels) | | False | True | Undetermined |
|---|---|---|---|---|---|
| | Accuracy | F1 (weighted) | F1 | F1 | F1 |
| **Baseline** | | | | | |
| BERT-base | 0.67 | 0.68 | 0.38 | 0.79 | 0.27 |
| **Regular Inputs** | | | | | |
| RoBERTa-base | 0.67 | 0.69 | 0.51 | 0.77 | 0.34 |
| RoBERTa+24 | 0.68 | 0.66 | 0.33 | 0.80 | 0.0 |
| **Injected Hypothesis Input** | | | | | |
| RoBERTa-base | 0.71 | 0.69 | 0.32 | 0.82 | 0.20 |
| RoBERTa+24 | 0.65 | 0.66 | 0.42 | 0.77 | 0.0 |
| **BERT Hidden Layer Concatenating (On Regular Inputs)** | | | | | |
| BERT last 2 layers | 0.67 | 0.69 | 0.40 | 0.78 | 0.45 |
| BERT last 4 layers | 0.69 | 0.69 | 0.38 | 0.80 | 0.28 |
| **Tweaked Attention (On Regular Inputs)** | | | | | |
| RoBERTa-x2 | 0.65 | 0.68 | 0.46 | 0.76 | 0.35 |
| RoBERTa-x100 | 0.74 | 0.63 | 0.85 | 0 | 0 |
| **Word Importance with Attention Mask (WIAM) (Proposed and now Implemented)** | | | | | |
| RoBERTa-base (WIAM) | 0.72 | 0.63 | 0.02 | 0.84 | 0.28 |

*Table 1: Shown are the results of our experiments. We show that adding more encoders in front of the usual BERT-base architecture tends to affect the learning process. Moreover, injecting the hypothesis into each premise turn helps to a certain extent, giving the best result of the accuracy of 0.71 and F1 of 0.69. Tweaking attention by a factor of 2 leads to reasonable results; however, tweaking attention by a factor of 100 hampers the learning process again. Lastly, RoBERTa-base (WIAM) continues to give comparable results to the other tweaked attention models, however we see that the learning is inhibited during the training process.*

Our best model results in a weighted F1 of 0.69 and an accuracy of 71%. We see that adding more encoder layers (and hence more self-attention) like RoBERTa+24 does not seem to help significantly. This is probably due to the fact that adding new layers will end up introducing weights with only the initialized values. Hence the error computed towards the end of the network would be higher in magnitude. Therefore, the larger errors tend to disturb the already well-established weights in the beginning layers corresponding to the first 12 encoders.

Tweaking the attention layers was done under the premise that an increase in the factor of attention would improve performance. We observe for RoBERTa x2 that the model is able to learn for the first three epochs and produces reasonable results comparable to our other architectures.

In the case of RoBERTa x100, we observe that although the model trains for 3 epochs, the validation loss does not go down. Therefore, we do not see much learning happening here. The model just predicts the majority class.

On the other hand, RoBERTa-WIAM performs much better than RoBERTa-x100 (even though we see during the training process, that the learning pattern is similar to that of RoBERTa-x-100). We are able to achieve multi-class prediction here opposed to just the majority class.

**4.1 Error Analysis:**

We conduct a qualitative analysis of 50 error instances by RoBERTa-base. We see that the majority of errors are made due to negation (70%). Out of these errors due to negation, in 35% of them, the model predicts False instead of True.

Temporal and Spatial errors are made 10% of the time each. These are basically errors made due to incorrect judgment of time and positional information, respectively.

We also have instances where we hypothesize that the error is due to the presence of several named entities. Several named entities in the premise and hypothesis confuse the model. Such errors contribute to about 8% of the total errors.

Lastly, we have 2% of the errors made due to extensive numerical data present in the premise and hypothesis. The model is unable to create the appropriate reasoning when there is a heavy amount of numbers in the input text.

We provide examples for each of the errors in the appendix. Percentages of errors made have also been illustrated in table 2. This analysis has been heavily inspired by *(Sanagavarapu et al., 2022)*.

## 5. Individual Contribution

**Shivam Mathur**

- Manual and GPT-3 data generation (35 manual samples and 350 GPT-3 samples)
- Worked on understanding the attention mechanism in the encoder architecture of BERT.
- Devised attention tweaking ideas (used in Tweaked Attention Experiments), seeking inspiration from K-BERT's attention modification mechanism.
- Worked closely with Sakthi to conceptualize RoBERTa-x2 and RoBERTa-x100
- Experimentation of RoBERTa-base and RoBERTa+24 (Injected Hypothesis Input)
- Performed qualitative error analysis on the best-performing model.

**Md Nayem Uddin**

- Manual and GPT-3 data creation (35 manual samples and 350 GPT-3 samples)
- Manually distribute the different labels among the generated samples from GPT-3
- Experimented with the BERT baseline model.
- Implemented the BERT model variation tweaking the hidden output layers.

**Saurav Anchlia**

- Manual and GPT-3 data generation (35 manual samples and 350 GPT-3 samples)
- Wrote python script to preprocess data
- Manually distributed the different labels among the generated samples from GPT-3
- Experimented with BERT base models using the outputs from hidden states.

**Sakthi Ganesh**

- Manual and GPT-3 data generation (35 manual samples and 350 GPT-3 samples)
- Performed data preprocessing and cleansing
- Experimented with RoBERTa Attention Score x 2 and Attention Score x 100 and evaluated model performance
- Worked on model explainability using LIME to understand the wrong predictions.
- Implemented the proposed mechanism - feeding normalized word importance along with attention mask and evaluated model performance.

## 6. Conclusions

We see that GPT3 (when prompted appropriately) is able to generate decent-quality data instances for our various logical reasoning tasks. Further, i. encoder-based LLMs like BERT and RoBERTa can have their attention layers tweaked, and their performance can be analyzed. We also run experiments by ii. adding new layers of encoders, iii. formatting inputs differently, and iv. utilizing different layers of the BERT model and analyzing performance. We have also implemented a word importance-based attention mask model that relies on a Soft Attention Mask mechanism facilitated by the LIME explainability tool. Lastly, we run an error analysis to understand where the LLM fails to predict correct labels.

## 7. References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

[2] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, pages 6000–6010.

[4] W. Liu, P. Zhou, Z. Zhao, Z. Wang, Q. Ju, H. Deng, and P. Wang, "K-BERT: Enabling language representation with knowledge graph," in AAAI, vol. 34, no. 03, 2020, pp. 2901–2908.

[5] Krishna Sanagavarapu, Jathin Singaraju, Anusha Kakileti, Anirudh Kaza, Aaron Mathews, Helen Li, Nathan Brito, and Eduardo Blanco. 2022. Disentangling Indirect Answers to Yes-No Questions in Real Conversations. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4677–4695, Seattle, United States. Association for Computational Linguistics.

[6] Samuel R. Bowman, Gabor Angeli, Christopher Potts, & Christopher D. Manning (2015). A large annotated corpus for learning natural language inference. *CoRR, abs/1508.05326.*

## Appendix

### A.1. More Details on BERT

BERT has 12 attention heads (and 12 encoders). The input given to it is of a maximum of 512 tokens, and it outputs an embedding of 768 size for each instance.

BERT was trained on 2 tasks specifically- Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

In the case of BERT, 15% of the tokens are masked (or hidden in the input), and the model is trained to predict these tokens. During the training process, 80% of tokens are replaced by <*mask*>, 10% are replaced by a random token, and the remaining 10% are left intact.

In NSP, the input is of the format *sentence-1 [SEP] sentence-2*. The model is being trained to predict if sentence-1 follows sentence-2. The paper talks about an auxiliary NSP loss that helps in this process. We do not delve into the details of this NSP loss in this report.

The architecture is termed bidirectional since the encoder is able to self-attend to the entire instance from start to end. As opposed to this, decoder-only based models (like GPT) do not look ahead of the current token during the decoding process. This is due to the autoregressive nature of the decoder.

For our baseline, we use the BERT-base uncased model available on hugging face.

## A.2. More Details on RoBERTa

### Dynamic Masking in RoBERTa as Opposed to Static Masking in BERT

In the case of BERT, the same instance was duplicated 10 times with different masks. The training was done for 40 epochs. This meant that the model saw the same instance with the same mask 4 times.

In the case of dynamic masking (which is done in RoBERTa), a masking pattern is generated on the fly whenever the instance is input into the model. Hence the chances of seeing the same input with the same mask reduce.

### Use of Full Sentences rather than Segments for NSP

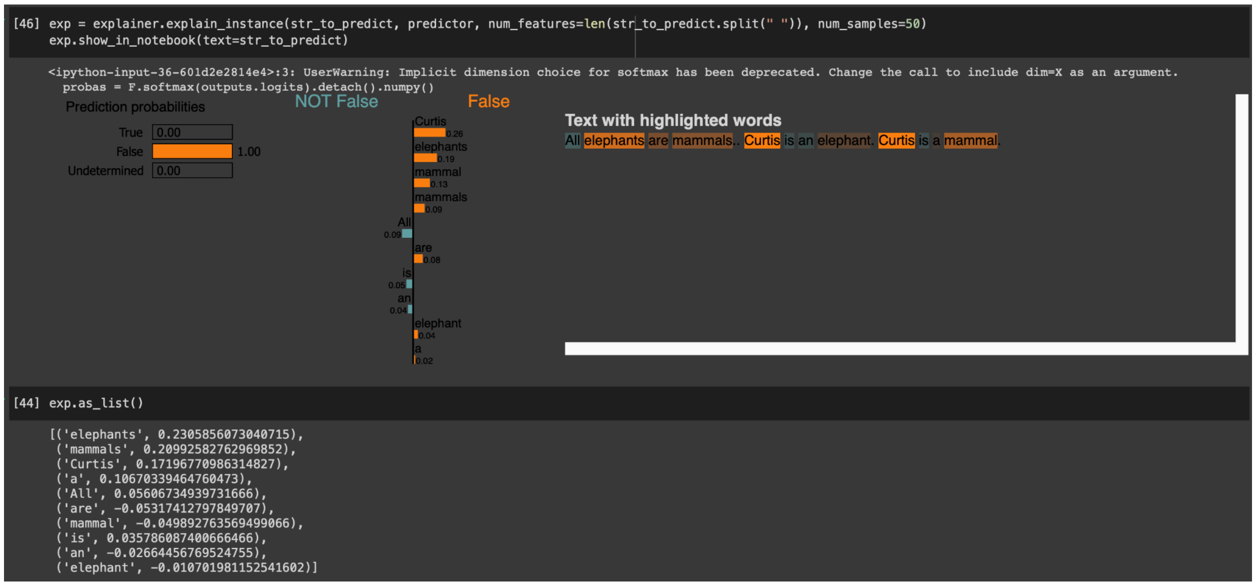
BERT uses a pair of two segments for an NSP task. A segment could be one or more sentences of natural language, each less than 512 tokens. The segment pairs can either be from the same document or from a different document.

In the case of RoBERTa, the strategy is to fill this 512 capacity for each input. During this process, document boundaries can be exceeded. A separator token is added for this reason. Further, the auxiliary NSP loss is removed, and the performance on downstream tasks shows improvements.

### Increasing batch size for pre-training RoBERTa

The authors increase the batch size of what was in the original BERT from 256 to 2000 and observe better results.

## A3. Using Word Importance along with Attention Mask

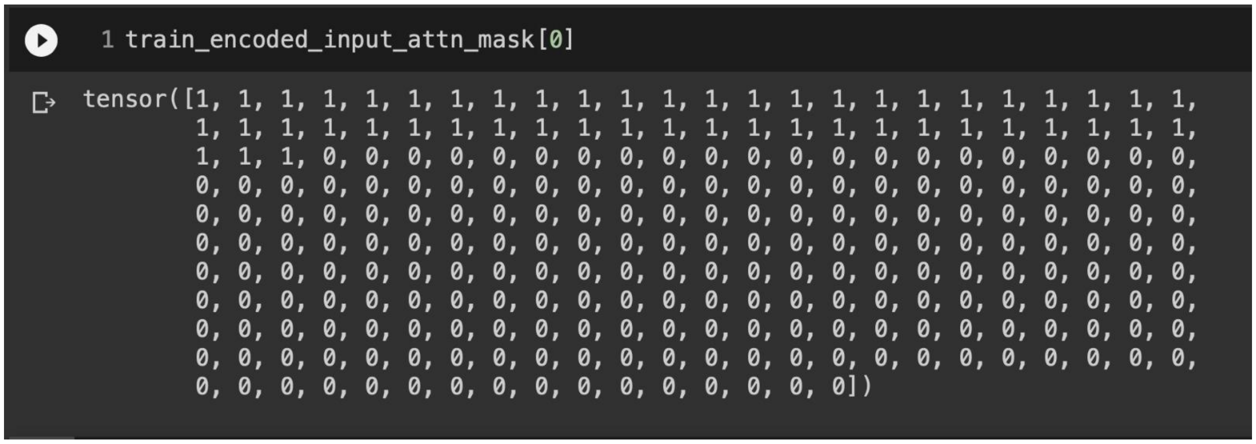


```
[46] exp = explainer.explain_instance(str_to_predict, predictor, num_features=len(str_to_predict.split(" ")), num_samples=50)
     exp.show_in_notebook(text=str_to_predict)

     <ipython-input-36-601d2e2814e4>:3: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
       probas = F.softmax(outputs.logits).detach().numpy()
```

All elephants are mammals.. Curtis is an elephant. Curtis is a mammal.

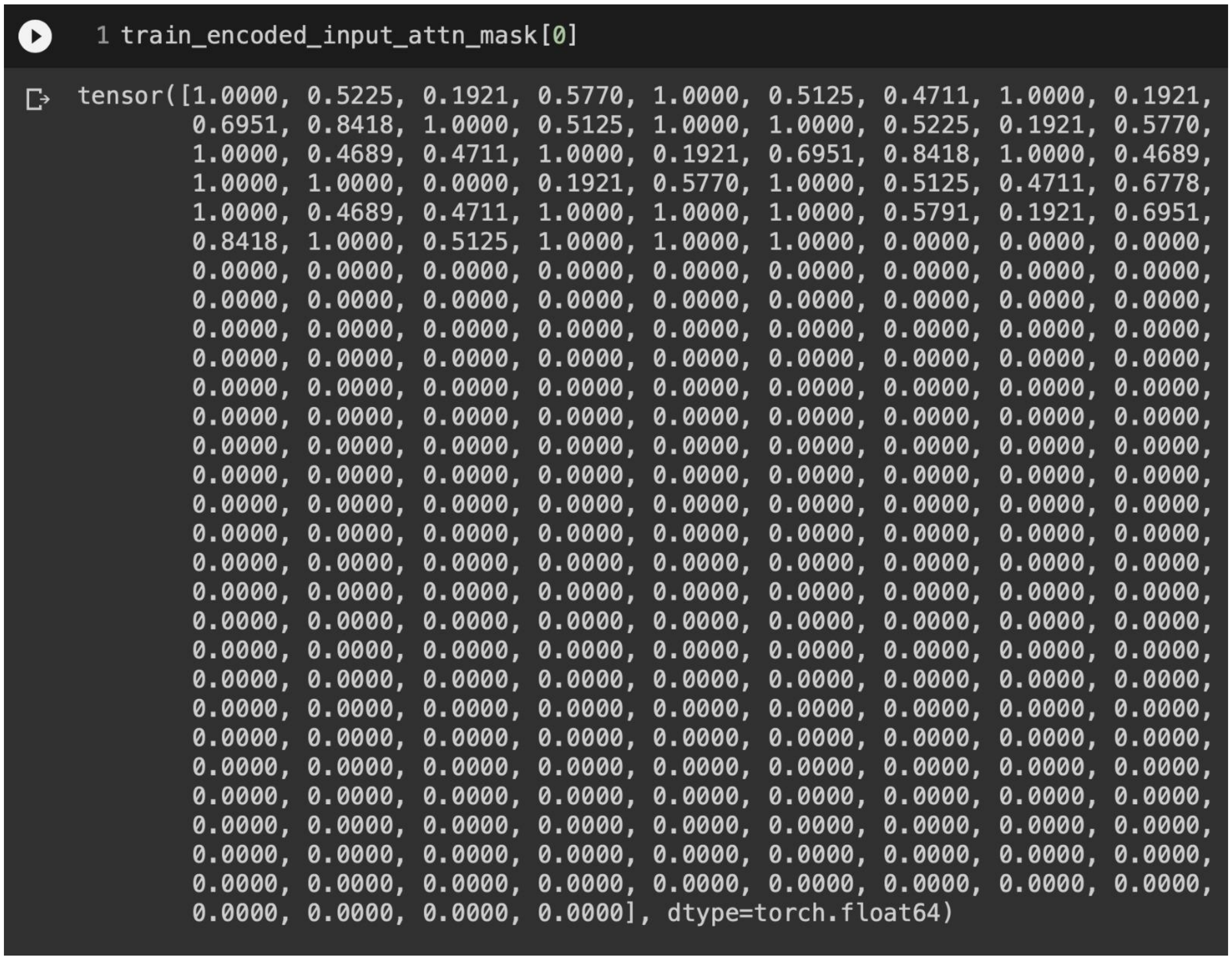
```
[44] exp.as_list()

     [('elephants', 0.2305856073040715),
      ('mammals', 0.20992582762969852),
      ('Curtis', 0.17196770986314827),
      ('a', 0.10670339464760473),
      ('All', 0.05606734939731666),
      ('are', -0.05317412797849707),
      ('mammal', -0.049892763569499066),
      ('is', 0.035786087400666466),
      ('an', -0.02664456769524755),
      ('elephant', -0.010701981152541602)]
```

*Figure 2: The LIME Explainer takes the sentence (premise + hypothesis), well-performing model, and its tokenizer as input and provides the importance of each word in the sentence for making a class prediction as the output, as shown in the diagram above.*



```
train_encoded_input_attn_mask[0]

tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

*Figure 3: Shows the old attention mask prior to adding word importance from LIME. Notice that values are discrete - either 0 or 1.*



```
train_encoded_input_attn_mask[0]
```

```
tensor([1.0000, 0.5225, 0.1921, 0.5770, 1.0000, 0.5125, 0.4711, 1.0000, 0.1921,
        0.6951, 0.8418, 1.0000, 0.5125, 1.0000, 1.0000, 0.5225, 0.1921, 0.5770,
        1.0000, 0.4689, 0.4711, 1.0000, 0.1921, 0.6951, 0.8418, 1.0000, 0.4689,
        1.0000, 1.0000, 0.0000, 0.1921, 0.5770, 1.0000, 0.5125, 0.4711, 0.6778,
        1.0000, 0.4689, 0.4711, 1.0000, 1.0000, 1.0000, 0.5791, 0.1921, 0.6951,
        0.8418, 1.0000, 0.5125, 1.0000, 1.0000, 1.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000, 0.0000], dtype=torch.float64)
```

*Figure 4: Shows the new attention mask after adding word importance from LIME. Notice that values are continuous, ranging from 0 to 1.*

**A.5. Error Analysis**

| Error Type | Gold | Prediction | Percentage |
|---|---|---|---|
| Negation (70%) | True | False | 34.3 |
| | False | True | 22.9 |
| | Undetermined | False | 20.0 |
| | True | Undetermined | 14.3 |
| | False | Undetermined | 5.7 |
| | Undetermined | True | 2.9 |
| Spatial Information (10%) | False | True | 20 |
| | True | False | 20 |
| | True | Undetermined | 20 |
| | Undetermined | False | 20 |
| | Undetermined | True | 20 |
| Temporality (10%) | False | True | 40 |
| | True | Undetermined | 40 |
| | Undetermined | True | 20 |
| Named Entity (8%) | True | Undetermined | 50 |
| | False | True | 25 |
| | Undetermined | False | 25 |
| Numeric Information (2%) | True | Undetermined | 100 |

*Table 2: We present an error analysis of 50 randomly picked error instances. We manually go over each instance and assign buckets to each. Most errors (70%) occur due to negation in either the premise or hypothesis. Out of these, the model tends to predict 'False' for 34% of instances that have a ground truth 'True'. Other error classes are shown with the percentage frequency of label misclassification. This table complements section 4.1 in the main report.*

**A.5. Examples of Error Types**

1. Negation: The premise and/or hypothesis contains a negation cue. (no, not, cannot, ‘t, etc.)

Premise: If I do not wake up, then I cannot go to work.. If I cannot go to work, then I will not get paid.
Hypothesis: If I do not wake up, I will not get paid.

Gold: True
Predicted: False

2. Spatial Information: The premise and/or hypothesis contains positional information about objects, their presence and absence in locations.

E.g. Premise: Person A put a laptop, an ipad in his bag while going out.. Today he forgot to put either the laptop or the ipad in the bag.
Hypothesis: Both the laptop and the ipad is missing in the bag today.

Gold: False
Predicted: True

3. Temporality: The premise and/or hypothesis contains the notion of time ( in hours, weeks, etc.)

E.g. Premise: I will play cricket on weekends if there is no rain.. I played cricket last weekend.
Hypothesis: There was rain last weekend.

Gold: False
Predicted: True

4. Named Entities: The premise and/or hypothesis contains Named Entities.

E.g. Premise: Apples are grown at Washington state in the United States.
Hypothesis: Therefore, somewhere in the United States Apples are grown.

Gold: True
Predicted: Undetermined

5. Numeric Information:  The premise and/or hypothesis contains numbers. These could be percentages, financial terms etc.

E.g. Premise: If I have 1 million dollars, I will donate half of it.. If I have 10 million dollars, I will donate 75% of it.. Either I have 1 million dollars or 10 million dollars.
Hypothesis: Therefore I will donate 500 thousand dollars or 7.5 million dollars.

Gold: True
Predicted: Undetermined