# Assignment 2

RELEASE DATE: 02/24/2025

DUE DATE: 03/17/2025 11:59pm on Canvas

LaTeX Template: https://www.overleaf.com/read/jjnbvkjxzmtr#d18ae9

Name: First-Name Last-Name UIN: 000000000

*This assignment consists of two parts: a writing section and a programming section. For the writing section, please use the provided LaTeX template to prepare your solutions and remember to fill in your name and UIN. For the programming section, please follow the instructions carefully.*

*When answering problems, please provide a* ***detailed and step-by-step explanation*** *to justify your solutions.*

*Discussions with others on course materials and assignment solutions are encouraged, and the use of AI tools as assistance is permitted. However, you must ensure that* ***the final solutions are written in your own words****. It is your responsibility to avoid excessive similarity to others' work. Additionally, please clearly* ***indicate any parts where AI tools were used*** *as assistance.*

*If you have any question, please send an email to* csce638-ta-25s@list.tamu.edu

## 1 Tokenization with Byte-Pair Encoding [15pts]

Considering the following words and their frequencies

| Word | Frequency | Initial Vocabulary |
|---|---|---|
| t a g g e d [/w] | 5 times | t a g e d<br>i n j u s [/w] |
| t a g g i n g [/w] | 2 times | |
| j u d g e d [/w] | 4 times | |
| s i n g i n g [/w] | 4 times | |

where [/w] represents the end of word. Initially, the vocabulary contains 11 character-level subwords. Using the Byte-Pair Encoding method introduced in the lecture, construct the vocabulary until it reaches 20 subwords. For each new subword added, please explain the reason.

**Solution:**

Please enter your solution here.

## 2 Visualization of Positional Encoding (Programming) [15pts]

`CSCE638-S25-HW2-2.ipynb`: Colab Notebook

Please use your *@tamu.edu* email to access the Colab Notebook. Copy the Colab Notebook to your drive and make the changes. The notebook has marked blocks where you need to code.

### ========= TODO : START ========= ###
### ========= TODO : END ========= ###

**Please copy and paste your code** (between `TODO:START` and `TODO:END`) **as part of the solution.** You can use the Minted package for code highlighting. Here is one example:

```
def hello_world():
    print("Hello World!")
```

## 2.1 Construing Positional Encoding [5pts]

In the lecture, we introduce the *sinusoidal positional encoding*

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

where $pos$ is the position index and $d_{model}$ is the embedding dimension.

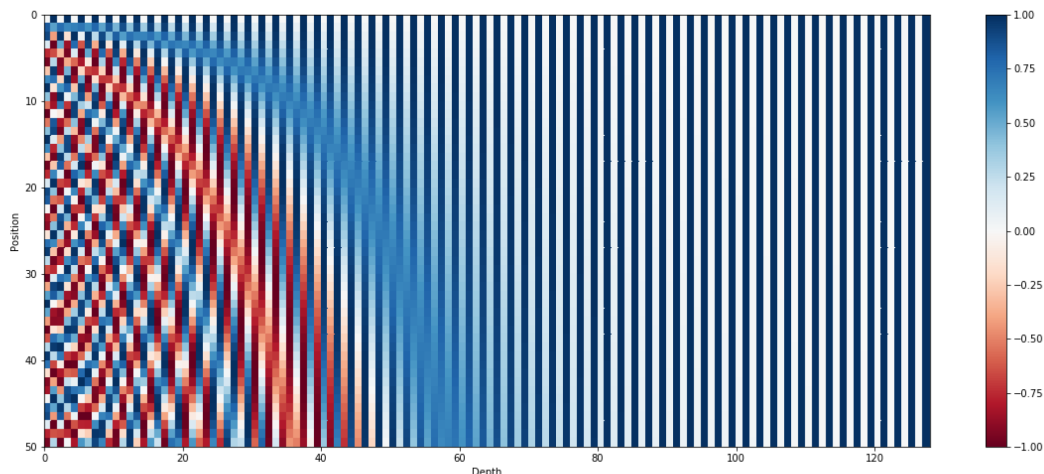
Please follow the instructions and implement the related parts to construct a `max_pos` × `d_model` positional encoding matrix, where the $i$-th row is the corresponding positional embedding with `d_model` dimensions for the position index $i$. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_construct_positional_encoding`.

**Solution:**

Please enter your solution here.

## 2.2 Visualizing Positional Encoding [5pts]

Please use `matplotlib` to visualize the positional encoding, with the x-axis representing the dimension and the y-axis representing the position index. Please set `max_pos` to 200 and set `d_model` to 250. You can refer to the following figure for an example. It is fine if you use different colors for visualization.



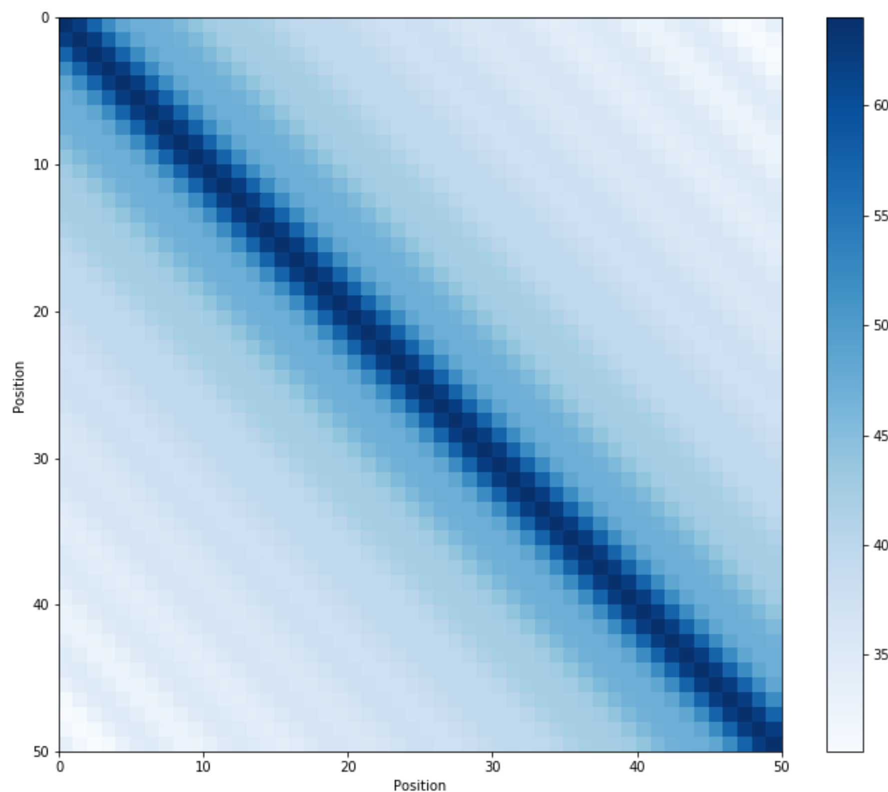
Copy and paste your code (between `TODO:START` and `TODO:END`) and include the visualization figure.

**Solution:**

Please enter your solution here.

## 2.3 Visualizing Similarity Between Positional Encoding [5pts]

Please use `matplotlib` to visualize the similar between positional encoding, with the x-axis and the y-axis representing the position index separately. The value of (x, y) should be the cosine similarity of encoding at position x and encoding at position y. Please set `max_pos` to 200 and set `d_model` to 250. That means the similarity matrix should be a $200 \times 200$ matrix. You can refer to the following figure for an example. It is fine if you use different colors for visualization.




Copy and paste your code (between `TODO:START` and `TODO:END`) and include the visualization figure.

**Solution:**

Please enter your solution here.

# 3 Advanced Text Classification (Programming) [35pts]

We will build classifiers for the Yelp Review dataset by training a convolutional network and fine-tuning the pre-trained BERT-base model. Yelp Review dataset is a 3-class classification task, where the input will be a short review and the output will be 0/1/2 corresponding to negative/okay/positive. Unlike the simplified pipeline used in Assignment 1, this time we will implement a more comprehensive pipeline.

`CSCE638-S25-HW2-3.ipynb`: Colab Notebook

`glove.6B.50d.txt`: Data

`train.json`: Data

`valid.json`: Data

`test.json`: Data

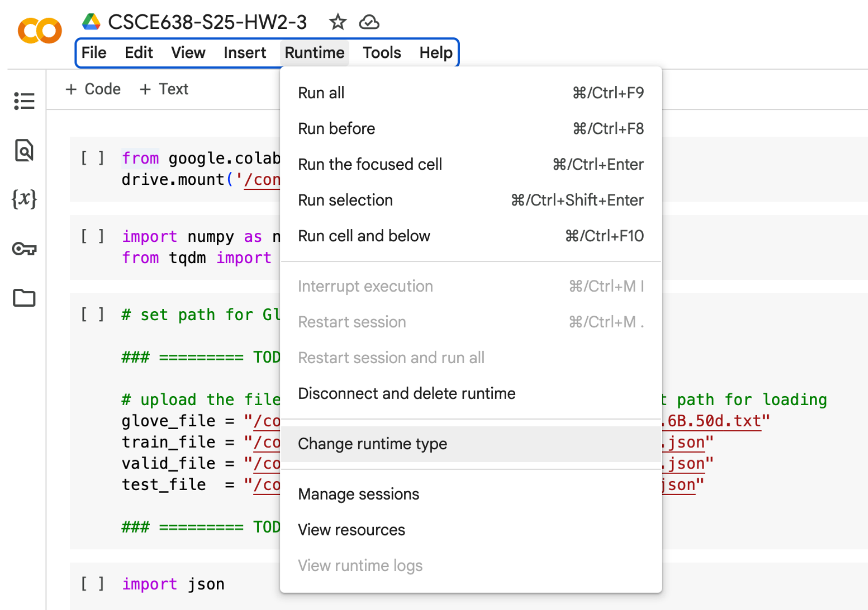
Please use your *@tamu.edu* email to access the Colab Notebook. Copy the Colab Notebook to your drive and make the changes. The notebook has marked blocks where you need to code.

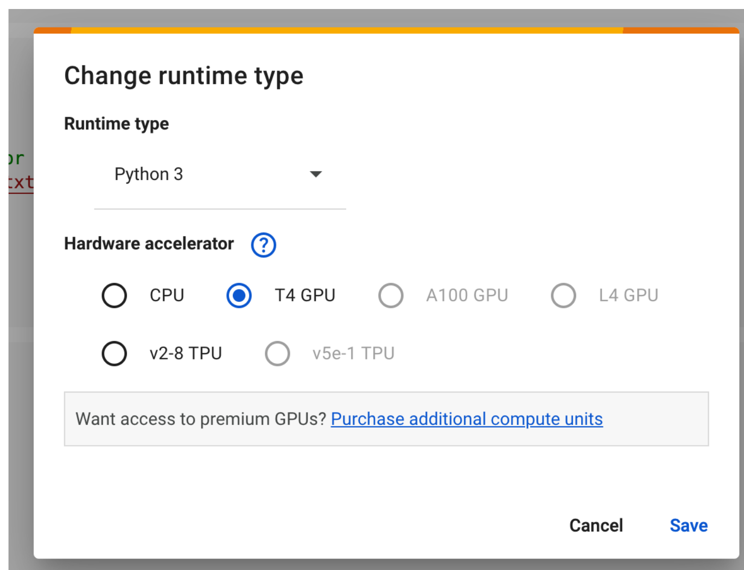
$$\#\#\# ========= TODO : START ========= \#\#\#$$
$$\#\#\# ========= TODO : \ \ END \ \ ========= \#\#\#$$

**Please copy and paste your code** (between `TODO:START` and `TODO:END`) **as part of the solution.** You can use the Minted package for code highlighting. Here is one example:

```python
def hello_world():
    print("Hello World!")
```

For this problem, you might have to *change the Colab runtime type* to enable GPU computation. Please choose the T4 GPU.






### 3.1 Building Vocabulary [5pts]

We are going to build a vocabulary from the *tokenized* corpus (train+valid+test) based the word frequency. There are two main components for a vocabulary: `idx2word` and `word2idx`. `idx2word` is a list to map each index in the vocabulary to the corresponding word, while `word2idx` is a dictionary to map each word in the vocabulary to the corresponding index.

Index 0 and index 1 are two special tokens. Index 0 corresponds to the padding token `[PAD]`, which will be used later to ensure all texts have the same length. Index 1 represents the unknown token `[UNK]`, which will be used for out-of-vocabulary tokens.

Please follow the instructions and implement the related parts to build the vocabulary. A word is added to the vocabulary only if its occurrence in the corpus meets or exceeds a specified threshold. Additionally, words should be sorted by frequency (from high to low), except for `[PAD]` and `[UNK]`, which remain in fixed positions. For example, the word `idx2word[2]` should appear more frequently than the word `idx2word[3]`.

Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_vocab`.

**Solution:**

Please enter your solution here.

### 3.2 Converting Text to Index [3pts]

We will convert each tokenized text to a fixed-length index vector, where each index corresponds to a token based on the vocabulary mapping. The index vector needs a fixed length because it is easier for batch calculation. However, since texts may vary in length, we will append the padding tokens `[PAD]` to ensure they are of equal length.

For example, a text "*I love cats*" should be converted to a vector $[v_1, v_2, v_3]$ first, where $v_1$ is the

index of *I*, $v_2$ is the index of *love*, and $v_3$ is the index of *cats*. If we set the maximum sequence length to 10, the final index vector should be a 10-dimensional vector $[v_1, v_2, v_3, 0, 0, 0, 0, 0, 0, 0]$, where 0 is the index of `[PAD]`.

Please follow the instructions and implement the related parts to convert texts to index vectors. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_text2idx`.

**Solution:**

Please enter your solution here.

### 3.3 Preparing Convolutional Network [8pts]

First, we need an *embedding layer* to convert each word index to the corresponding word embedding. You can learn how to use `torch.nn.Embedding` to implement it here.

We consider the following convolutional network

$$\text{output} = \text{Linear}(\text{ReLU}(\text{MaxPooling}([\text{CNN features}])))$$

where the CNN features can be obtained by applying several different sizes of CNN filters to word embeddings, as we discussed in the lecture.

We will use a list (`filter_sizes`) to indicate the sizes of filters and we will consider a certain number (`num_filter_per_size`) of filters for each size. For example, if `filter_sizes` is $[3, 4, 5]$ and `num_filter_per_size` is 10, it means we have 10 filters with size 3, 10 filters with size 4. and 10 filters with size 5. You will need to use `torch.nn.Conv1d` (link) and `torch.nn.MaxPool1d` (link) for the implementation. For more information and details of CNN, you can check the original TextCNN paper and this implementation. (Notice that the implementation is a bit different!)

Please follow the instructions and implement the related parts to prepare the model. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_model`.

**Solution:**

Please enter your solution here.

### 3.4 GloVe Initialization [4pts]

The word embeddings created by `torch.nn.Embedding` is randomly initialized. We would like to replace some embeddings with GloVe embeddings if the words match.

Specifically, we have to check if a word in the vocabulary has a GloVe embedding. If yes, replace the corresponding embedding (randomly initialized) with the GloVe one. You can use `torch.Tensor.copy_` (link) to copy the vector. Meanwhile, you have to count how many words in the vocabulary have corresponding GloVe embeddings, denoted as `n_match` in the code.

Please follow the instructions and implement the related parts to initialize embeddings. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_initialize_glove`.

**Solution:**

Please enter your solution here.

### 3.5 Training and Testing [2pts]

Please follow the instructions and implement the related parts to complete the training loop. Notice that you may have to use `Tensor.cuda` or `torch.device` to enable GPU computation. You can learn more about them here and here. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the cell output and the final accuracy.

**Solution:**

Please enter your solution here.

### 3.6 Adjusting Vocabulary Size [2pts]

Please change the threshold for building the vocabulary to 1 (using all the words) and run the training pipeline again. Report the final accuracy. Compared to the accuracy from Problem 3.5, what are your findings and what are the possible explanations?

**Solution:**

Please enter your solution here.

### 3.7 BERT Tokenization [3pts]

We are going to use Hugging Face's `Transformers` package to play with the pre-trained BERT model. Read the document of `transformers.BertTokenizer` (link) and follow the instructions and implement the related parts to load pre-trained BERT tokenizer and get corresponding *index vectors* and *attention mask*. Specifically, please load `google-bert/bert-base-uncased` for BERT. One simple examples can be found here

Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_bert_tokenize`.

**Solution:**

Please enter your solution here.

### 3.8 More on BERT Tokenization [4pts]

Check the document of tokenizer (link) and use `tokenizer.convert_ids_to_tokens` to convert the index vectors back to normal text. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_text_idxs_to_text`.

Additionally, output the corresponding tokens for index **100**, **101**, **102**, and **103** and explain the purpose of those tokens.

**Solution:**

Please enter your solution here.

### 3.9 Fine-Tuning BERT [4pts]

Read the document of `transformers.BertForSequenceClassification` (link) and follow the instructions and implement the fune-tuning loop. Specifically, please load `google-bert/bert-base-uncased` for BERT.

Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the cell output and the final accuracy. Compared to CNN's result, what are your findings and explanations?

**Solution:**

Please enter your solution here.

## 4 Sequence-to-Sequence (Programming) [35pts]

We will develop sequence-to-sequence models for the Quora Question Pairs dataset. Given one Quora question, the sequence-to-sequence model is trained to generate a question with a similar meaning.

`CSCE638-S25-HW2-4.ipynb`: Colab Notebook

`glove.6B.50d.txt`: Data

`train-q.json`: Data

`valid-q.json`: Data

`test-q.json`: Data

Please use your *@tamu.edu* email to access the Colab Notebook. Copy the Colab Notebook to your drive and make the changes. The notebook has marked blocks where you need to code.

### ========= *TODO* : *START* ========= ###
### ========= *TODO* : *END* ========= ###

**Please copy and paste your code** (between `TODO:START` and `TODO:END`) **as part of the solution.** You can use the Minted package for code highlighting. Here is one example:

```
def hello_world():
    print("Hello World!")
```

Similar to Problem 3, you might have to *change the Colab runtime type* to enable GPU computation. Please choose the T4 GPU.

### 4.1 Building Vocabulary [1pts]

Similar to Problem 3, we are going to build a vocabulary from the *tokenized* corpus based the word frequency. We follow the same sorting rules as Problem 3.1.

One difference here is that we have 4 special tokens. Index 0 corresponds to the padding token `[PAD]`, which will be used later to ensure all texts have the same length. Index 1 denotes the begin-of-sentence token `[BOS]`, while index 2 denotes the end-of-sentence token `[EOS]`. Index 3 represents the unknown token `[UNK]`, which will be used for out-of-vocabulary tokens.

Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_vocab`.

**Solution:**

Please enter your solution here.

### 4.2 Converting Text to Index [2pts]

Similar to Problem 3, we will convert each tokenized text to a fixed-length index vector, where each index corresponds to a token based on the vocabulary mapping. Different from Problem 3.2, we have to add `[BOS]` and `[EOS]` to the begin and end of texts before adding `[PAD]`. For example, a text "*I love cats*" should be converted to a vector $[v_1, v_2, v_3]$ first, where $v_1$ is the index of *I*, $v_2$ is the index of *love*, and $v_3$ is the index of *cats*. Then, we add the corresponding indices of `[BOS]` and `[EOS]` to obtain $[1, v_1, v_2, v_3, 2]$. If we set the maximum sequence length to 10, the final index vector should be a 10-dimensional vector $[1, v_1, v_2, v_3, 2, 0, 0, 0, 0, 0]$, where 0 is the index of `[PAD]`.

Please follow the instructions and implement the related parts to convert texts to index vectors. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_text2idx`.

**Solution:**

Please enter your solution here.

### 4.3 Preparing Dataset and DataLoader [1pts]

For a sequence-to-sequence task, we have to prepare three different sequences: (1) encoder input, (2) decoder input, and (3) decoder output. Since we are doing generation, the decoder input and the decoder output should have a index shift. For example, given a 10-dimensional index vector $[1, v_1, v_2, v_3, 2, 0, 0, 0, 0, 0]$, the decoder intput should be a 9-dimensional vector $[1, v_1, v_2, v_3, 2, 0, 0, 0, 0]$ and the decoder output should be a 9-dimensional vector $[v_1, v_2, v_3, 2, 0, 0, 0, 0, 0]$.

Please follow the instructions and implement the related parts to prepare the dataset and dataloader. Copy and paste your code (between `TODO:START` and `TODO:END`).

**Solution:**

Please enter your solution here.

### 4.4 Preparing LSTM Encoder [7pts]

We consider a one-layer unidirectional LSTM with hidden size being `hidden_size` as the encoder. You can check here to learn how to use `torch.nn.LSTM`. Similar to Problem 3, we need an *embedding layer* to convert each word index to the corresponding word embedding. You can learn how to use `torch.nn.Embedding` to implement it here. Particularly, the LSTM encoder should take an input index vector and return the encoder output embeddings, the hidden state, and the cell state. You can refer to here for more information.

Please follow the instructions and implement the related parts to prepare the LSTM encoder. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_encoder`.

**Solution:**

Please enter your solution here.

### 4.5 Preparing LSTM Decoder [7pts]

We consider a one-layer unidirectional LSTM with hidden size being `hidden_size` as the decoder. The LSTM decoder has two parts different from the LSTM encoder: (1) The input of the decoder should be an input index vector as well as the hidden state and the cell state from the encoder. (2) The output of decoder is *logits*, which can be calculated as follows

$$\text{Logits} = \text{Linear}(\text{Decoder Output})$$

The logits have the same dimension to the size of vocabulary for calculating the cross entropy loss.

$$\mathcal{L}_{CE} = \text{CrossEntropyLoss}(\text{Logits}, \text{Ground Truth Word})$$

Again, you can check here for reference.

Please follow the instructions and implement the related parts to prepare the LSTM decoder. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_decoder`.

**Solution:**

Please enter your solution here.

### 4.6 Preparing LSTM Seq2Seq Model [8pts]

Now, we combine the LSTM encoder and the LSTM decoder togather as the Seq2Seq model. The Seq2Seq requires a `generate` function for inference. During inference, only the encoder input is available, and the output must be generated word by word. To accomplish this, you will implement a loop for iterative generation. Starting from `[BOS]`, use the decoder to predict the next token, append it to the previously generated tokens, and repeatedly feed the updated sequence into the decoder until the maximum length `max_len` is reached. Check here for reference.

When generating the next word, you may need to apply `torch.nn.Softmax` (link) to the logits to get the distribution over the vocabulary and use `torch.multinomial` (link) to sample a word from the distribution.

Please follow the instructions and implement the related parts to prepare the LSTM Seq2Seq model. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_seq2seq`.

**Solution:**

Please enter your solution here.

### 4.7 GloVe Initialization [1pts]

Similar to Problem 3. The word embeddings created by `torch.nn.Embedding` is randomly initialized. We would like to replace some embeddings with GloVe embeddings if the words match. Remember to replace the embeddings for both the encoder and the decoder.

Please follow the instructions and implement the related parts to initialize embeddings. Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_initialize_glove`.

**Solution:**

Please enter your solution here.

### 4.8 Training [4pts]

Please follow the instructions and implement the related parts to complete the training loop. Notice that you may have to use `Tensor.cuda` or `torch.device` to enable GPU computation. You can learn more about them here and here.

When computing loss, we have to ignore `[PAD]` as we don't want it to affect the generation process. You can check here to learn how to ignore a particular index when computing loss.

You may adjust the learning rate, the number of epochs, or any other things to optimize performance. However, keep in mind that due to resource constraints, the model is designed to be trained on only a limited number of examples. As a result, the validation loss may not decrease significantly, and the quality of the generated text may be lower than expected. As long as the final generated texts are understandable to humans, your answer will be accepted.

Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the cell output.

**Solution:**

Please enter your solution here.

### 4.9 Converting Index Vector to Text [3pts]

Please follow the instructions and implement the related parts to convert index vectors to texts. For each index vector, discard all the indices after the first `[EOS]` and convert all the remaining indices to words. For example, given the index vector $[1, 13, 20, 2, 0, 2, 7]$, the first `[EOS]` appears in the fourth position (2). Therefore, we should convert $[1, 13, 20, 2]$ to text, which will be "`[BOS] do best`".

Copy and paste your code (between `TODO:START` and `TODO:END`) as well as the output of `test_idxs2texts`.

**Solution:**

Please enter your solution here.

### 4.10 Testing [1pts]

Please run the final testing. Keep it in mind that your answer will be accepted as long as the final generated texts are understandable to humans. It is fine if the generated texts do not match the inputs very well. Copy and paste the cell output.

**Solution:**

Please enter your solution here.

## Submission Instructions

You have to upload a `.zip` file to Canvas, which contains the following:

- `submission.pdf`: The `.pdf` file generated by the LaTeX template.
- `submission2.py`: Please export the Colab Notebook for problem 2 to a `.py` file by clicking "File" → "Download" → "Download .py"
- `submission2.ipynb`: Please export the Colab Notebook for problem 2 to a `.ipynb` file by clicking "File" → "Download" → "Download .ipynb"
- `submission3.py`: Please export the Colab Notebook for problem 3 to a `.py` file by clicking "File" → "Download" → "Download .py"
- `submission3.ipynb`: Please export the Colab Notebook for problem 3 to a `.ipynb` file by clicking "File" → "Download" → "Download .ipynb"
- `submission4.py`: Please export the Colab Notebook for problem 4 to a `.py` file by clicking "File" → "Download" → "Download .py"
- `submission4.ipynb`: Please export the Colab Notebook for problem 4 to a `.ipynb` file by clicking "File" → "Download" → "Download .ipynb"