Assignment

Instructions:

This assignment consists of 2 tasks. Task 1 focuses on your data processing and the ability to use NLP based libraries for various kinds of sentence processing. Task 2 tests your ability to build and train neural networks such as the Transformer to approach a certain problem. Your results from Task 1 shall be prerequisite to your evaluation of Task 2.

Language of implementation should be Python 3.6 or up.

Final Submission Link: Submit Here

Task 1

Oftentimes it gets important to automate evaluation tasks while dealing with Language Models. Most popular among all being creation of data (fake data). In this task you will be creating two different versions from the same dataset.
Once you have generated the required files for both the versions zip them to a folder named '<your_full_name>_task1.zip>'. You need to submit this folder for evaluation of Task 1.

Dataset: The **Multi-Genre Natural Language Inference (MultiNLI)** corpus is a crowd-sourced collection of 433k sentence pairs annotated with textual entailment information. The Hugging Face 'GLUE' repository will give you direct access to the training data.

Guidelines to process the data:

- Load the dataset using the **'datasets'** library with the following piece of code:

```
from datasets import load_dataset
dataset = load_dataset('glue', 'mnli')
```

- From the **train** split of the data, extract the **first 5000** sentences from the **'premise (string)'** column and use them further.

Task Definition: You need to process and create the new datasets as per the following rules.

1. **Negation ('not')**: Insert the negation marker 'not' after the first auxiliary verb in the original true target sentence to generate a new distractor sentence. If the sentence already contains the negation marker 'not', we instead remove it. 'Not' Negation aims to detect whether a sentence embedding model is misled by the negation of a sentence relation caused by adding the word 'not'.

   Important Guidelines:

- Use the NLTK in-built functionalities 'word_tokenize' for tokenization and 'pos_tag' for tagging the parts of speech.
- Transfer the processed sentences to a **CSV** file with columns

  ['index', 'real_sentence', 'fake_sentence']

  .
- Code in a notebook and title it '<your_full_name>_negation.ipynb'. Name the generated csv file '<your_full_name>_negation.csv'. Put both the files in the folder for Task 1.

Examples:

- Real: The movie was slow.
  Fake: Them movie was **not** slow.
- Real: She did **not** eat ice-cream.
  Fake: She did eat ice-cream..

2. **Antonyms**: First find a sentence containing an adjective and then replace the adjective with its antonym to obtain sentence pairs. Note that these antonym pairs generally bear a derivational connection. In cases where there is a set of antonyms for a particular word, you will select the antonym that comes alphabetically first using WordNet.

   Important Guidelines:

   - Use the NLTK in-built functionalities 'word_tokenize' for tokenization and 'pos_tag' for tagging the parts of speech.
   - If a sentence has more than one adjective, substitute all of them in the same sentence.
   - Use 'wordnet' from 'nltk.corpus' to generate the list of unique antonyms and then choose the first one alphabetically to substitute in the original sentence.
   - If a word does not have an antonym available, keep the word as it is in the sentence.
   - Transfer the processed sentences to a **CSV** file with columns

     ['index', 'real_sentence', 'fake_sentence']

     .
   - Code in a notebook and title it '<your_full_name>_antonyms.ipynb'. Name the generated csv file '<your_full_name>_antonyms.csv'. Put both the files in the folder for Task 1.

   Examples:

   - Real: The movie was **slow**.
     Fake: Them movie was **fast**.
   - Real: The idea was **good** and **interesting**.
     Fake: The idea was **bad** and **boring**.

Task 2

In this task you need to build a Transformer architecture that trains to solve a binary classification problem in NLP. You will be working with a small amount of fabricated data and report your scores on the training set itself.

Data:

sentences = ['you won a billion dollars , great work !',
'click here for cs685 midterm answers',
'read important cs685 news',
'send me your bank account info asap']

labels = $[1, 1, 0, 1]$

Task Definition:

To solve the above binary classification problem, build a transformer with 2 encoder blocks.
Use Sinusoidal Positional Encoding to embed positional information in the positional encoding layer. Record precision, recall and F1 score.

Parameters:

- number of encoder blocks/ number of layers = 2
- number of heads (multi-head attention) = 2
- dimension of the embeddings = 96 (using Spacy)
- feed forward layer dimension = 32
- dropout rate = 0.1
- max length of sentence (if using padding) = 10
- batch size = 4
- number of epochs = 10

Guidelines

- You may use either PyTorch or TensorFlow for your code. If you do code in both the frameworks send us both submissions in the same zip file.
- Use Spacy to compute the input embeddings with the following piece of code:

```
import spacy
nlp = spacy.load('en_core_web_sm')
x = nlp(<input>)
```

Report

Submit a zip file titled '<your_full_name>_transformer.zip' containing the following:

- Your well commented code in an **.ipynb** format and title it depending on your framework, i.e. PyTorch or TensorFlow, 'transformer_<framework>.ipynb'.
- A csv file, with columns ['precision','recall','F1'] depicting your results. Again based on your framework name it '<framework>_results.csv'
- If you have used both PyTorch and TensorFlow, then submit both with the respective naming convention in the same zip file.

References and Help

- The Illustrated Transformer
- Transformer Architecture: The Positional Encoding
- A Gentle Introduction to Positional Encoding In Transformer Models