CSE 556: Natural Language Processing Assignment 3

Deadline - 6 April 2025, Sunday (11:59 pm IST)

Instructions

1. **Plagiarism** will be very strictly dealt with, and institute policy will apply.
2. The assignment is time consuming so please start timely. Each member is expected to contribute to a task individually and should be aware of all other tasks.
3. You have to submit a single .zip file named **A3-{Group No.}.zip**. Example: **A3-24.zip**.
4. **Viva:** During the demo, some questions related to the concepts covered in this task and your implementation will be asked. The marks for the viva will not be based on group they will be assigned individually.
5. **Important Note:** Evaluation scores for all tasks on the test samples contribute to the final grade, so students should optimize their models and training strategies to achieve the best possible performance while strictly using the GPU resources available on Google Colab or Kaggle.

1 Task - Implement Transformer from Scratch 25 Marks

1.1 Task Description

The goal of this task is to implement a **Transformer** model from scratch for language modeling using the **Shakespearean dataset**. The model should be trained to generate text in Shakespeare's style.

1.2 Dataset Description

You will be provided with `shakespeare_train.txt`, and `shakespeare_dev.txt` files containing sentences in the following format:

```
First Citizen: Before we proceed any further, hear me speak.
All: Speak, speak.
First Citizen: You are all resolved rather to die than to famish?
All: Resolved.
First Citizen: First, you know Caius Marcius is the chief enemy to the people.
All: We know't, we know't.
```

1.3 Code Implementation

In this task, you will be provided with a code template and must complete the defined functions by implementing the following:

- Data Preprocessing and Tokenization

- Self-Attention and Multi-Head Attention
- Causal Masking for Autoregressive Generation
- Transformer Model and Components

Essential Transformer concepts, including:

- Positional Encoding
- Layer Normalization
- Residual Connections
- Scaled Dot-Product Attention
- Feed-Forward Networks
- Softmax Normalization
- Weight Initialization
- Dropout Regularization

must be implemented and integrated into the model.

1.4 Model Training

The training procedure must include the following steps:

- Implement the training loop to optimize the model using loss computation and backpropagation.
- Track and record the training and validation losses across epochs.
- Try generating future tokens based on a given context after every epoch to assess the quality of the trained model.
- Save the trained model for future inference.
- Include plots of the training and validation losses for the model in the report.

1.5 Evaluation Criteria

The model's performance will be assessed using **perplexity**. If padding tokens are used during training, they must be excluded from the perplexity calculation. Failure to adhere to this requirement will result in a score of zero.

1.6 Testing and Model Inference

During the demo, a `test.txt` file with sample sentences will be provided. You must implement a function that:

- Loads the pretrained model.
- Reads and processes the input from `test.txt`.
- Generates a fixed number of tokens for each given sentence based on the provided context.
- Calculates the perplexity score.

1.7 Deliverables

You must submit the following:

- Implementation code for preprocessing, model training, and inference in a single file named `task1.py` or `task1.ipynb`.
- The saved model for Task 1.
- A detailed report including:
  - Explanation of preprocessing steps.
  - Model architecture and hyperparameters used.
  - Training and validation loss plots.

2 Task - Claim Normalization 30 Marks

Claim Normalization is a novel task that involves transforming complex and noisy social media posts into clear and structured claims, referred to as **normalized claims**. This task will be evaluated using the **CLAN dataset**, which comprises of real-world social media posts along with their corresponding normalized claims. You can refer to the paper here for more details.

2.1 Dataset Description

Students will be provided with the `CLAN_data.csv` file, which contains social media posts along with their corresponding normalized claims, annotated by a professional fact-checker. The dataset should be divided into training, validation, and test sets in a 70-15-15 ratio for model training and evaluation.

2.2 Preprocessing

Students need to preprocess the social media posts before claim normalization. The following steps should be included:

- **Expand contractions and abbreviations**: Replace common contractions and abbreviations with their expanded forms (e.g., *he'll* → *he will*, *she's* → *she is*, *Gov.* → *Governor*, *Feb.* → *February*, *VP* → *Vice President*, *ETA* → *Estimated Time of Arrival*).
- **Clean text**: Remove links, special characters, and extra whitespace while converting text to lowercase.

2.3 Model Training

You are required to train both **BART** and **T5** models by fine-tuning them on the provided dataset. The choice of **BART** and **T5** variants is left to your discretion, with the goal of achieving the best possible results while optimizing for limited GPU resources available in **Google Colab** or **Kaggle Notebooks**.

2.4 Evaluation

The task will be evaluated using **ROUGE-L**, **BLEU-4**, and **BERTScore** to measure lexical and semantic similarity. The best-performing model must be saved for inference.

2.5 Testing and Model Inference

A `test.csv` file similar to `CLAN_data.csv` file will be provided during the demo. Students must implement a function that:

- Loads the trained model.
- Processes the `test.csv` file.
- Generates normalized claims.
- Computes all the evaluation metrics.

2.6 Deliverables

You must submit the following:

- Implementation code for preprocessing, model training, and inference in a single file named `task2.py/task2.ipynb`.
- The best-performing saved model.
- A detailed report including:
  - An explanation of preprocessing steps.
  - Model architecture and hyperparameters used.
  - Training and validation loss plots for the models.
  - Evaluation metrics on the test set for both models.
  - A comparative analysis of the performance of both models.
  - Discussion on resource constraints and how they influenced model selection.

3 Task: Multimodal Sarcasm Explanation (MuSE) 45 Marks

3.1 Task Overview

The objective of this task is to develop a **Multimodal Sarcasm Explanation (MuSE)** model. For a deeper understanding of the task and methodology, refer to the following paper: Research Paper

3.2 Dataset Description

The task utilizes the **MORE+ dataset**, which comprises sarcastic posts sourced from social media platforms such as Twitter, Instagram, and Tumblr. Each sample consists of:

- An image
- A corresponding textual caption
- A manually annotated sarcasm explanation
- A manually annotated sarcasm target

To simplify implementation, a preprocessed dataset will be provided, including:

- All images in the dataset
- Train, and validation pickle files containing:
  - Image descriptions (e.g. : **D_train.pkl**)
  - Detected objects (e.g. : **O_train.pkl**)
- Data files (e.g. : **train_df.tsv**) with fields: post ID (pid), text, explanation, and sarcasm target

Access the dataset here: MORE+ Dataset

3.3 Model Requirements

Students must implement the **TURBO model** without using **Knowledge Graphs** (KG) and **Graph Convolutional Networks** (GCN). The model should:

- Extract high-level image features using a **Vision Transformer**
- Concatenate token sequences following Section 4.3 of the referenced paper
- Incorporate the **sarcasm target** into the explanation generation process
- Be based on the **BART base model**
- Implement a **Shared Fusion Mechanism**

3.4 Key Considerations

1. Ensure that the model effectively focuses on relevant tokens within the shared fusion mechanism.
2. The weights of the shared fusion mechanism should remain trainable.

3.5 Evaluation Metrics

The model's performance will be assessed using the following metrics, as specified in the paper:

- ROUGE (R-L, R-1, R-2)
- BLEU (B-1, B-2, B-3, B-4)
- METEOR
- BERTScore

3.6 Testing and Model Inference

A `test.tsv` file, formatted similarly to `train_df.tsv`, along with `D_test.pkl` and `O_test.pkl` files, will be provided for evaluation. Students are required to implement a function that:

- Loads the trained model.
- Processes these files.
- Generates and stores sarcasm explanations in an output file.
- Computes and reports all evaluation metrics.

3.7 Deliverables

You must submit the following:

- Implementation code for preprocessing, model training, and inference in a single file named `task3.py/task3.ipynb`.
- Model checkpoints saved after each epoch.
- A detailed report including:
  - An explanation of preprocessing steps.
  - Model architecture and hyperparameters used.
  - Training loss reported at each epoch.
  - Evaluation metrics on the validation set after each epoch.
  - Few generated sarcasm explanations on the validation set for the last epoch.