Fill-in-the-Blank Prediction using LSTM Networks

Due Date: 17 November 2024

Objective

The goal of this assignment is to apply LSTM networks to fill in missing words within sentences. Using the RACE dataset, you will construct fill-in-the-blank questions by removing a random word from the latter half of each sentence. Two LSTM networks will then predict the missing word based on the context:

- **Forward LSTM:** Uses the first half of the sentence (before the blank) to predict the missing word.
- **Backward LSTM:** Uses the reversed second half of the sentence to predict the blank from the context that follows it.

In case the forward LSTM and backward LSTM give different predictions, you must find a way to select the better answer for filling in the blank and explain your choice in the report.

Dataset

- **Dataset:** RACE Dataset
- **Description:** The RACE dataset contains reading comprehension passages and associated questions. You will use it to create sentences, remove words to form blanks, and use these sentences as inputs to your LSTM models.
- TO LOAD:
  from datasets import load_dataset

  dataset = load_dataset("race", "all")

Tasks

1. Data Preprocessing

1. Select sentences from the RACE dataset.
2. For each sentence, remove a random word in the latter half to create a fill-in-the-blank question.
3. Split the sentence into two parts:
    - Part A: The first half of the sentence, ending just before the blank.
    - Part B: The second half of the sentence, starting just after the blank and reversed.

2. LSTM Network Design

1. Model 1 (Forward LSTM): Pass Part A to an LSTM to predict the missing word.
2. Model 2 (Backward LSTM): Reverse Part B and pass it to a separate LSTM to predict the missing word.

3. Model Implementation

1. You may choose a pretrained LSTM-based language model (e.g., from Hugging Face) and finetune it on your fill-in-the-blank sentences.
2. Alternatively, you can build your own LSTM model(s) from scratch using PyTorch or Keras.
    - Ensure that both models (Forward and Backward) share the same output vocabulary, so they predict within the same word set.

4. Training and Evaluation

1. Split the dataset into training and validation sets.
2. Measure the accuracy of each model by comparing the predicted word to the actual missing word.
3. Report the performance of each model on the validation set.

5. Fine-tuning and Hyperparameter Tuning

- Experiment with different hyperparameters, such as batch size, learning rate, number of LSTM layers, and embedding dimensions.
- Record your hyperparameters and performance metrics for each experiment.

6. Prediction Handling

- If the forward and backward LSTM models suggest different words for the blank, you must devise a way to select the better option. Explain your decision-making process and provide justification for the approach in your report.

Report

Your report should include:

1. Data Preprocessing Details: Describe how you processed the dataset, created blanks, and prepared input for each LSTM model.
2. Model Architecture: Explain the architecture of each LSTM model, including any layers added for preprocessing (e.g., embedding layers) and the output layers.
3. Training Details: Include training and validation performance, hyperparameters, and any tuning you performed.
4. Observations: Reflect on the accuracy of each model, differences between the forward and backward predictions, and any challenges faced.

Submission

1. Code: Submit a well-documented Jupyter notebook or Python script that:
    - Processes the RACE dataset,
    - Defines and trains both LSTM models,
    - Outputs model performance and evaluation metrics.
2. Report: Submit a report in PDF format that includes all the details mentioned above.
3. Results: Include sample outputs from both models and any additional insights.

Bonus

1. Ensemble Prediction: Combine predictions from both models to enhance the final word prediction accuracy. Describe your approach in the report.
2. Exploration of Alternate Architectures: Experiment with alternate RNN-based architectures, such as bidirectional LSTMs, GRUs, or transformer-based models, and compare their performance with the standard LSTMs.

Grading Criteria

1. Data Preprocessing and Quality of Fill-in-the-Blank Questions (20%)
2. Model Implementation and Architecture (40%)
3. Performance and Evaluation (20%)
4. Report Quality (20%)

Guidelines

1. Clearly document each step in your code, especially around data preprocessing and model implementation.
2. Test your code with a small subset of data first to ensure it's working as expected before scaling up.
3. Be creative with your model design and tuning.

Good luck, and remember to justify your decisions clearly!