Assignment 2: Neural Language Modeling

Course: Introduction to NLP

Deadline: 10 February | 23:59

General Instructions

1. The assignment must be implemented in Python.
2. The assignment must be done using PyTorch (you are supposed to use standard functions only). Usage of other frameworks will not be accepted.
3. You are allowed to use AI tools or any internet resources as long as your implementation is in accordance with the assignment guidelines.
4. A single .zip file needs to be uploaded to the Moodle Course Portal.
5. Please start early since no extension to the announced deadline would be possible.

1 Language Models for Next Word Prediction (40 marks)

Next Word Prediction (NWP) involves predicting the most probable word that follows a given sequence of words. Mathematically, given a sequence of words $w_1, w_2, \ldots, w_{n-1}$, the goal is to determine the next word w_n such that:

$$w_n = \arg\max_{w \in V} P(w|w_1, w_2, \ldots, w_{n-1})$$

where V is the vocabulary and $P(w|w_1, w_2, \ldots, w_{n-1})$ is the conditional probability of w given the preceding words.

1.1 Feed Forward Neural Network Language Model (10 marks)

A Feed Forward Neural Network (FFNN) language model is designed to predict the next word in a sequence. The model processes a fixed-size window of preceding words and uses fully connected layers with activation functions to output probabilities for all words in the vocabulary. The predicted words are based on the top k probabilities.

Implementation Requirement: Use n-grams with $n = 3$ and $n = 5$ for training the FFNN.

1.2 Vanilla Recurrent Neural Network Language Model (15 marks)

A Vanilla Recurrent Neural Network (RNN) language model predicts the next word in a sequence by maintaining a hidden state that captures the sequential dependencies of the preceding words. The RNN processes input word embeddings iteratively and outputs probabilities for the next word at each step. The model predicts words based on the top k probabilities.

1.3 Long Short-Term Memory Language Model (15 marks)

A Long Short-Term Memory (LSTM) language model extends the Vanilla RNN by addressing the vanishing gradient problem. It employs memory cells and gating mechanisms (input, forget, and output gates) to capture long-term dependencies in the sequence. The LSTM outputs probabilities for the next word, with predictions based on the top k probabilities.

2 Corpus

The following corpora have been given to you for training:

- Pride and Prejudice corpus (1,24,970 words)
- Ulysses Corpus (2,68,117 words)

Please download the corpus files from this link.

3 Analysis (20 marks)

For each corpus, construct a test set by randomly selecting 1,000 sentences, ensuring this set is excluded from training. Perform the following steps for each language model (LM):

1. **Perplexity Calculation:**
   - Compute the perplexity score for each sentence in the *Pride and Prejudice* and *Ulysses* corpora using the trained LMs.
   - Calculate the average perplexity score across the training corpus for each model.
2. **Reporting:**
   - Document the perplexity scores for all sentences in the training set.
   - Report the perplexity scores for the test sentences in the same manner.
3. **Comparison and Ranking:**
   - Compare the perplexity scores of the current LMs with those from Assignment 1.
   - Rank the models based on their performance.
   - Provide a detailed analysis of the results, highlighting differences in model performance and potential reasons for these variations.
4. **Observations:**
   - Which model performs better for longer sentences? Why?
   - How does the choice of n-gram size affect the performance of the FFNN model?

Submission Format

Zip the following files into one archive and submit it through the Moodle course portal. The filename should be `<roll number>_assignment2.zip`, for example, `2021101034_assignment2.zip`.

- **Source Code**
  - `generator.py`:
    - **Usage:** `generator.py <lm_type> <corpus_path> <k>`
    - **lm_type:** Specifies the type of language model. The options are:
      - `-f`: Feedforward Neural Network (FFNN)
      - `-r`: Recurrent Neural Network (RNN)
      - `-l`: Long Short-Term Memory (LSTM)
    - **corpus_path**: Denotes the file path of the respective dataset.
    - **k:** Denotes the number of candidates for the next word to be printed.
  - **Output:** On running the file, the script provides a prompt that:
    - Asks for a sentence as input.
    - Outputs the most probable next word of the sentence along with its probability score using the specified language model.
    - Example:

      ```
      python3 generator.py -f ./corpus.txt 3
      Input sentence: An apple a day keeps the doctor
      Output:
      away 0.4
      happy 0.2
      fresh 0.1
      ```
  - You are encouraged to write modular code and include other intermediary files in your submission.
- **Pretrained Models**
  - .pt files containing your pretrained models or the drive link to them in your README.
- **Report (PDF)**
  - Your analysis of the results.
- **README**

  - Instructions on how to execute the file, load the pretrained model, implementation assumptions etc.

Ensure that all necessary files are included in the zip archive.

Grading

Evaluation will be individual and will be based on your viva, report and submitted code review. You are expected to walk us through your code, explain your experiments, and report. You will primarily be graded based on your conceptual understanding.

Implementation: 40 marks

Analysis: 20 marks

Viva during Evaluation: 40 marks

Resources

1. Neural Networks and Deep Learning
2. How to Create a Neural Network (and Train it to Identify Doodles)
3. RNNs and LSTMs
4. Understanding LSTM Networks
5. You can also refer to other resources, including lecture slides!