SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.040 Natural Language Processing, Summer 2019 Mini Project

Due 17 June 2019, 5pm

This project will be graded by Richard Sun

Introduction Language models are very useful for a wide range of applications, e.g., speech recognition and machine translation. Consider a sentence consisting of words $x_1, x_2, \ldots, x_m$, where m is the length of the sentence, the goal of language modeling is to model the probability of the sentence, where $m \geq 1$, $x_i \in V$ and V is the vocabulary of the corpus:

$$p(x_1, x_2, \ldots, x_m)$$

In this project, we are going to explore both statistical language model and neural language model on the Wikitext-2 datasets.

Statistical Language Model A simple way is to view words as independent random variables (i.e., zero-th order Markovian assumption). The joint probability can be written as:

$$p(x_1, x_2, \ldots, x_m) = \prod_{i=1}^{m} p(x_i)$$

However, this model ignores the word order information, to account for which, under the first-order Markovian assumption, the joint probability can be written as:

$$p(x_0, x_1, x_2, \ldots, x_m) = \prod_{i=1}^{m} p(x_i \mid x_{i-1})$$

Under the second-order Markovian assumption, the joint probability can be written as:

$$p(x_{-1}, x_0, x_1, x_2, \ldots, x_m) = \prod_{i=1}^{m} p(x_i \mid x_{i-2}, x_{i-1})$$

Similar to what we did in HMM, we will assume that $x_{-1} = START, x_0 = START, x_m = STOP$ in this definition, where $START, STOP$ are special symbols referring to the start and the end of a sentence.

Parameter estimation Let's use $count(u)$ to denote the number of times the unigram u appears in the corpus, use $count(v, u)$ to denote the number of times the bigram v, u appears in the corpus, and $count(w, v, u)$ the times the trigram w, v, u appears in the corpus, $u \in V \cup STOP$ and $w, v \in V \cup START$. And the parameters of the unigram, bigram and trigram models can be obtained using maximum likelihood estimation (MLE).

In the unigram model, the parameters can be estimated as:

$$p(u) = \frac{count(u)}{c}$$

, where c is the total number of words in the corpus.

In the bigram model, the parameters can be estimated as:

$$p(u \mid v) = \frac{count(v, u)}{count(v)}$$

In the trigram model, the parameters can be estimated as:

$$p(u \mid w, v) = \frac{count(w, v, u)}{count(w, v)}$$

Smoothing the parameters It is likely that many parameters of bigram and trigram models will be 0 because the relevant bigrams and trigrams involved do not appear in the corpus. If you don't have a way to handle these 0 probabilities, all the sentences that include such bigrams or trigrams will have probabilities of 0.

We'll use a Add-k Smoothing method to fix this problem, the smoothed parameter can be estimated as:

$$p_{add-k}(u) = \frac{count(u) + k}{c + k|V^*|}$$

$$p_{add-k}(u \mid v) = \frac{count(v, u) + k}{count(v) + k|V^*|}$$

$$p_{add-k}(u \mid w, v) = \frac{count(w, v, u) + k}{count(w, v) + k|V^*|}$$

where $k \in (0, 1)$ is the parameter of this approach, and $|V^*|$ is the size of the vocabulary V^*,here $V^* = V \cup STOP$. One way to choose the value of k is by optimizing the perplexity of the development set, namely to choose the value that minimizes the perplexity.

Perplexity Given a test set D' consisting of sentences $X^{(1)}, X^{(2)}, \ldots, X^{(|D'|)}$, each sentence $X^{(j)}$ consists of words $x_1^{(j)}, x_2^{(j)}, \ldots, x_{n_j}^{(j)}$, we can measure the probability of each sentence s_i, and the quality of the language model would be the probability it assigns to the entire set of test sentences, namely:

$$\prod_{j}^{D'} p(X^{(j)})$$

Let's define average log2 probability as:

$$l = \frac{1}{c'} \sum_{j=1}^{|D'|} log_2 p(X^{(j)})$$

c' is the total number of words in the test set, D' is the number of sentences. And the perplexity is defined as:

$$perplexity = 2^{-l}$$

The lower the perplexity, the better the language model.

Task 1 (4 points) Remove the empty lines in the datasets, convert all the texts to lower cases, then compute counts of unigrams, bigrams, trigrams of the train corpus in the file "wiki.train.tokens". Do not take the START and STOP symbols into consideration.

- List numbers of unique unigrams, bigrams and trigrams respectively.
- List 10 most frequent unigrams, bigrams and trigrams as well as their counts.

Task 2 (4 points) Estimate the parameters for the bigram and trigram models through maximum-likelihood estimation respectively, compute the parameter for each n-gram in the file "ngram.txt", list down the n-grams that have 0 probability.

Take the START and STOP symbols into consideration. For example, given a sentence "I like NLP", in a bigram model, we need to pad it as "START I like NLP STOP", in a trigram model, we need to pad it as "START START I like NLP STOP".

Task 3 (6 points) Use the Add-k smoothing method to smooth parameters of bigram and trigram models respectively, choose the parameter k from the set 0.1, 0.3, 0.5, 0.7, 0.9 on the development set for each model. Compute the smoothed parameters of n-grams in the file "ngram.txt". The development data is in the file "wiki.valid.tokens".

Task 4 (4 points) Use the smoothed bigram and trigram models to compute the perplexity of the test set in the file "wiki.test.tokens" respectively. Which model has a lower perplexity?

Neural Language Model Using the chain rule, the probability of a sentence consisting of words $x_1, x_2, ..., x_n$ can be represented as:

$$p(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} p(x_t \mid x_{t-1}, ..., x_1)$$

Assume that we can use a hidden vector $h_t \in R^d$ of a recurrent neural network (RNN) to record the history information of words:

$$h_t = RNN(x_t, h_{t-1})$$

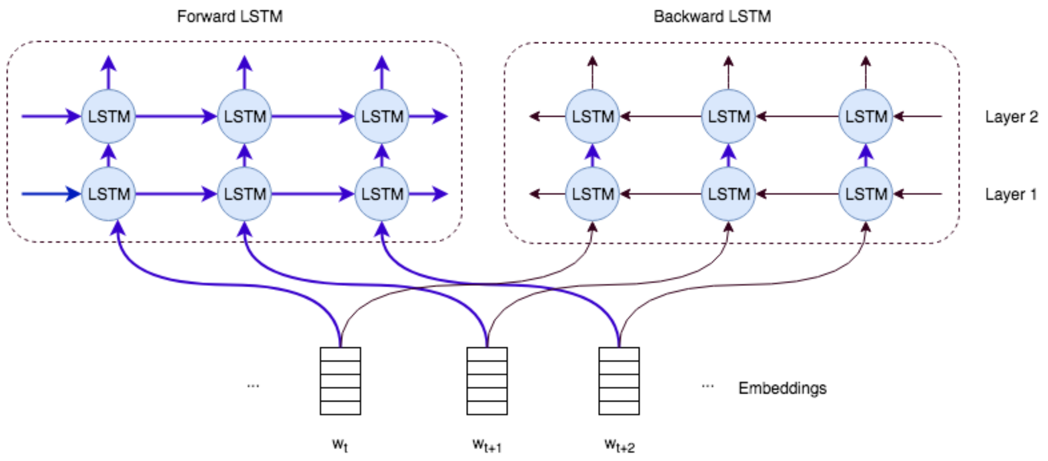

Figure 1: 2-layer Bidirectionl LSTM Language Model Architecture

The conditional probability of word x_{t+1} can be parameterized as:

$$p(x_{t+1} \mid x_t, x_{t-1}, ..., x_1) \propto exp(f(w_{x_{t+1}} h_t))$$

d is the dimension size of the hidden layer, $|V|$ is the size of the vocabulary. f is a fully-connected layer, where $w \in R^{|V| \times d}$ are the parameters, $w_{x_{t+1}}$ is the parameter in the row that corresponds to the index of x_{t+1} in the vocabulary, the bias is omitted.

Task 5 (12 points) We will create a LSTM language model, and train it on the Wikitext-2 dataset. The data generators(train_iter, valid_iter, test_iter) and the LSTM model(in the file "lstm_model.py") have been provided. The word embeddings together with the parameters in the LSTM model will be learned from scratch.

Our tasks:

- Complete the training and evaluating code, tune hyperparameters on the validation data, then compute the perplexity of the test data. The test perplexity should be below 150. (5 points)
- Visualize word embeddings trained by our language model as in Homework 1, try to find patterns, i.e., are similar words clustering? (2 points)
- Implement a 2-layer bidirectional LSTM language model as shown in Figure 1, train the language model from both directions, compute the perplexity of the test data for each direction. Note, the forward and backward LSTMs do not share parameters, and the outputs from the previous layer can be only passed to the next layer in the same direction. (5 points)

The START and STOP symbols have been added to the sentences in the generators, and the second dimension of the outputs of generators is the index of the batch.

Pytorch is required in this part. Do not make any changes to the provided code unless you are requested to do so.

Task 6 (Optional, 6 points) We can train our 2-layer bidirectional LSTM language model on a relatively large dataset wikitext-103 and explore some interesting characteristics of the model. Our tasks:

- Generate a piece of text ending with the STOP symbol but no more than 50 words given the START symbol, check whether it is fluent or not based on our tuition.
- Feed each sentence of the movie reviews in homework 1 to our language model, concatenate the outputs of forward and backward LSTMs as the contextualized embedding for each word in the sentence.
- Train the provided RNN Classifier in homework 1 based on the contextualized word embeddings of the train set, tune parameters on the development set and evaluate the performances on the test set.
- Consider "play", a highly polysemous word, find 20 sentences including "play", half of them refer to the verb, and half of them refer to the noun. Visualize the contextualized embeddings of "play" in the sentences, check whether the embeddings can capture the contextual information or not.

Requirements:

- This is an individual project.
- Complete answers and Python code in the "mini project.ipynb" file. Submit the file before the due on eDimension system.
- Write down names and IDs of students with whom you have discussed (if any).
- Follow the honor code strictly.