Big Data and Artificial Intelligence

Lab Assignment (TP) 3: Feed-Forward Language Model(FFLM)

Supervisor: Montmaure Antoine

Written by : *Othmane Cherai & Hamza Houbbane*

Instructor: *Son Vu*
Department: Département Informatique
Date: April 3, 2023

Contents

1 Part 1: N-Gram on WikiText

N-grams are a commonly used technique in natural language processing that involves analyzing the frequency of occurrence of sequences of words in a text. Specifically, an N-gram is a contiguous sequence of N items (typically words) within a text. By analyzing the frequency of N-grams in a text, researchers can gain insights into the patterns and structures of the language used in the text. In this lab report, we will explore the use of N-grams in the context of natural language processing. Specifically, we will examine how N-grams can be used to analyze the structure and content of text data, and we will discuss the various techniques and tools that are used to extract N-grams from text data. We will also explore some of the applications of N-grams in natural language processing, and we will discuss the strengths and limitations of this approach.

1.1 Mathematical Description

1.1.1 Discrete-Time Markov Chains

Markov chains are a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event or events. **Discrete-Time Markov Chains** are sequences of random variables $X_1, X_2, ...$ with the **Markov Proprety**:

$$P(X_{n+1} = x | X_1 = x_1, X_2 = x_2, ..., X_n = X_n) = P(X_{n+1} = x | X_n = x_n)$$

if both probabilities are well defined, *i.e* $P(X_1 = x_1, X_2 = x_2, ..., X_n = X_n) > 0$

1.1.2 N-Gram Model

Consider the task of computing $P(w|h)$, the probability of a word w given some history h. Suppose that history is some random sentence of length *n* composed of the words: $w_1, w_2, ..., w_n$. One way to estimate this probability is from relative frequency counts: take a large corpus, count the number of times we see the sequence $w_1, w_2, ..., w_n$ and count the number of times it is followed by w.

$$P(w|h) = \frac{C(w_1, w_2, ..., w_n, w)}{C(w_1, w_2, ..., w_n)}$$

However, for this method to be precise, we need a corpus that is humongously big. One common assumption of why the available corpora are not enough is because language is *creative*. The problem is already big enough when considering one word, as such, we will not consider the case where we will be trying to estimate whole sequences. Consider now the same above-mentioned sequence. We'll write the joint probability $P(X_1 = w_1, X_2 = w_2, ..., X_n = w_n)$ as $P(w_1, w_2, w_3, ..., w_n)$. The chain rule of probabilty gives:

$$P(w_{1:n}) = \Pi_{k=1}^{n} P(w_k | w_{1:k-1})$$

We can clearly see that the probability of a future unit by looking into *n-1* words in the past. Thus the considered **Markov** is satisfied. We say we are dealing with an **N-gram** when we are predicting the futures based on the *n-1* preceding words. The way to estimate probabilities

is called **Maximum Likelihood Estimation**. For the general case of MLE n-gram parameter estimation:

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}w_n)}{C(w_{n-N+1:n-1})}$$

1.2 Code Snippet

Most of the code features have been explained except fro the ones that are will be explained below:

- **Smoothing_k**: Smoothing is a technique used to adjust the probabilities of events based on the frequency of their occurrences in a given dataset. It is the operation of adding a constant value(k) to each count to avoid the problem of zero-count probabilities. This problem arises when we encounter a situation where an even tor outcome has never occurred in our observed data resulting in a probability estimate of zero. The drawbacks of this are **inability to further calculate probabilities** especially for the event that are dependent on the zero-probability event. Another problem that will arise is **Overfitting** as our model may learn to assign zero probabilities to events that were not observed in the training data.

$$P(w_t|w_{1:n}) = \frac{C(w_t, context) + k}{C(context) + |V|}$$

- **Logprob**: is as its name implies, measures the probability of a given event on a dataset.
- **Perplexity**: is a metric used to evaluate the performance of a language model. It measures how well a model can predict a sequence of words in a given dataset. A lower perplexity indicates a better language model performance. The difference between the test and train perplexity is an indicator of how well the language model generalizes to unseen data. If the difference is too big, with the test perplexity being too small compared to its counterpart, then the model is overfitting the train data

$$\begin{aligned} PPL(W) &= P(w_1, w_2, ..., w_n)^{-\frac{1}{n}} \\ &= -\frac{1}{n}\sum_{k=1}^{n} log(P(w_k|w_1, w_2, ..., w_{k-1})) \end{aligned} \tag{1}$$

- **generate_next(context, n)** Synthesises this to generate the different possibilities of the following n words of a certain context with the probability of each one of them.

2 Part 2: FFLM

FFLM are the implementation of word embeddings and Context C. In short, the embedding matrix becomes dependant on the context the word is in. Many of the operations that have been explained in previous labs are once again used here, such as mapping the vocabulary to a dictionary. As for every dataset, some features will be missing. In our case, words that are not in the training vocabulary will be given the special token **unk** . The network will learn embeddings of unknown tokens. However, if the corpus contains too many unknown words, we should consider using subwords or char-level representation to help the network better handle

out-of-vocabulary words. Although many of the items that will be presented below have been to some degree explained in the preceding part, they will be further detailed.

2.1 Perplexity and Entropy

As introduced in section1.2, **preplexity** is a way to evaluate n-gram models on a test set. A better n-gram model is one that assigns a higher probability to the test data and perplexity is a normalized version of the probability of the test set *(See formula above)*. Given that the perplexity is positively correlated to the quantity of processed data, it has to somehow be linked to **Entropy**, the latter being a measure of information. Despite the forthcoming analysis being out if the scope of this course, I believe It is necessary so as to understand the **Cross-Entropy Loss** introduced in 2.2.

2.1.1 One-Variable Entropy

Let X be a random variable(VA) ranging over whatever we are trying to predict which we will call χ. Let $p(X = x)$ noted as $p(x)$ be its particular probability function. Then **2-bit entropy** of X is:

$$H(X) = -\sum_{x \in \chi} p(x) log_2(p(x))$$

The most intuitive way to think about it is: **"Entropy is the lower bound on the number of bits it would take to encore a certain piece of information in the optimal coding scheme."**

2.1.2 Entropy Rate

Consider now the entropy of a *Sequence* $W = w_1, w_2, ..., w_n$ of a language *L*. It's entropy would then be:

$$H(W) = -\sum_{w_{1:n} \in L} p(w_{1:n}) logp(w_{1:n})$$

But to measure the entropy of a language, we need to consider countable sequences of infinite length. If we think of a language as a stochastic process(consider it a k-faces dice with k being the #L) that produces a sequence of words(*multinomial distribution)* then L's **Entropy rate** $H(L)$ is defined as:

$$\begin{aligned} H(L) &= lim_{n \to +\infty} \frac{1}{N} H(W) \\ &= -lim_{n \to +\infty} \frac{1}{N} \sum_{W \in L} p(W) logp(W) \end{aligned} \tag{2}$$

Furthermore, the Shannon-McMillan-Breiman theorem(p12) when considering the language stationary(which it is to some degree if we don't take into account the **unk** or are rare to find) and ergodic(a process whose proprieties can be deduced from a long enough process):

$$H(L) = -\frac{1}{n} logp(\prod_{i \leq n} w_i)$$

As such, long-enough sequences of words will contain in it many shorter sequences and that each of these shorter sequences will reoccur in the longer sequence to their probabilities.

A stochastic process is said to be **stationary** if the probabilities it assigns to a sequence are invariant with respect to shift in the time index. Markov models and hence n-grams are stationary.(If P_i is dependent on P_{i-1}, even when shifting the index by x, we still have P_{i+x} is dependent on P_{i-1+x}).

2.1.3 Cross Entropy

Cross-Entropy is useful when we don't know the actual probability distribution of p that generated some data(which is true for every text). It allows us to use *m*, which is a model of p(*i.e* approximation of p). It is defined as such:

$$H(p,m) = lim_{n\to+\infty} - \frac{1}{n}\sum_{w\in L} p(w_1,w_2,...,w_n)logm(w_1,w_2,...,w_n)$$

That is we draw a sequence according to *p* but sum the log of their probabilities according to *m*. Using the *Shanon-McMillan-Breiman* theorem we have:

$$H(p,m) = lim_{n\to+\infty} - \frac{1}{n}logm(w_1w_2...w_n)$$

That means that, as for entropy, we can estimate the cross-entropy of a model **m** on some probability p by taking a single sequence that is long enough instead of summing over all possible sequences. We also have for any model p: $H(P) \leq H(p,m)$. This means that the cross-entropy is an upper-bound on the entropy H(p). Hence, we can use some simplified model *m* to estimate the true entropy of a sequence of symbols drawn according to *p*. Cross-entropy is defined in the limit as the length of the observed word sequence goes to infinity. We will need an approximation to cross-entropy, relying on a sufficiently long sequence of fixed length(it is what we are doing in n-gram, beyond the n^{th} word is considered infinity). We then have:

$$H(W) = -\frac{1}{N}logP(w_1w_2...w_N)$$

2.1.4 Perplexity

The **Perplexity** of a model P on a sequence of words W is now formally defined as 2 raised to the power of cross-entropy:

$$\begin{aligned} Perplexity &= 2^{H(W)} \\ &= P(w_1w_2...w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1w_2...w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N}\frac{1}{P(w_i|w_1...w_{i-1})}} \end{aligned} \tag{3}$$

2.2 Code Snippet

CrossEntropyLoss is a loss function used in classification tasks when the output model is probability distribution over classes. It measures the difference between the predicted probability distribution and the true probability distribution of the labels. Its formula is:

```
loss(x, class) = −log(exp(x[class]) / (sum(exp(x)=)))
                = −x[class] + log(sum(exp(x)))
```

3 Summary

- Language models offer a way to assign a probability to a sentence or other sequences of words, and to predict a word from preceding words.
- n-grams are Markov models that estimate words from a fixed windows of previous words. N-Gram probabilities can be estimated by counting in a corpus.
- N-gram **language models** are evaluated using The perplexity.
- Smoothing algorithms provide a way to estimate the probability of n-grams