

Exercises on n -grams and fundamentals of language models

Ion Androutsopoulos, 2026–27

Submit (per person) a report for exercise 3 (max. 10 pages, PDF format). Explain briefly in the report the algorithms/methods that you used, how they were trained, how you tuned any hyper-parameters, how you prepared your data etc. Present in the report your experimental results and demos (e.g., screenshots) showing how your code works. Do not include code in the report but include a link to a Colab notebook (<https://colab.research.google.com/>) containing your code. During the examination of your submission, you may be asked to explain, modify, fill in, or improve any part of your submitted code. You are advised to use LLMs only for bugs you cannot fix, and for code review (to improve your final code), or not at all. For delayed submissions, one point will be subtracted per day of delay.

1. Calculate the entropy in slide 38 (e-mail messages). For $P(C) \rightarrow 0$, use L'Hôpital's rule.

Answer:

a) When the training examples comprise 200 spam και 600 ham messages, then:

$$P(C = 1) = \frac{200}{800} = \frac{1}{4}, P(C = 0) = \frac{600}{800} = \frac{3}{4}, \\ \log_2 \frac{1}{4} = -2, \log_2 \frac{3}{4} = \log_2 3 - \log_2 4 = 1.585 - 2 = -0.415$$

$$H(C) = \frac{1}{4} \cdot 2 + \frac{3}{4} \cdot 0.415 = 0.811$$

b) When the training examples comprise 400 spam και 400 ham messages, then:

$$P(C = 1) = \frac{400}{800} = \frac{1}{2}, P(C = 0) = \frac{400}{800} = \frac{1}{2}, \log_2 \frac{1}{2} = -1,$$

$$H(C) = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 1 = 1$$

c) When the training examples are spam, then:

$$P(C = 1) = 1, P(C = 0) = 0, \log_2 P(C = 1) = 0, \log_2 P(C = 0) \rightarrow -\infty$$

$$H(C) = -P(C = 1) \cdot \log_2 P(C = 1) - P(C = 0) \cdot \log_2 P(C = 0) =$$

$$= -1 \cdot 0 - \frac{\log_2 P(C = 0)}{\frac{1}{P(C = 0)}}$$

Using De L'Hôpital's rule for the second term, for $P(C = 0) \rightarrow 0^+$, we get:

$$H(C) \rightarrow -\frac{\frac{1}{P(C = 0) \ln 2}}{-\frac{1}{P(C = 0)^2}} = \frac{P(C = 0)}{\ln 2} = 0$$

Similarly, when all the training examples are ham, again $H(C) = 0$.

2. Show that a language model m_1 with perplexity r has the same perplexity (and cross-entropy) as a model m_2 that always assigns a uniform probability distribution (when

predicting the next word) to all the words of its vocabulary V_{m_2} of $|V_{m_2}| = r$ words, provided that the perplexity (and cross-entropy) of m_2 is estimated on a test corpus containing words all included in V_{m_2} .

Answer: Let's assume that the test corpus where we estimate the cross-entropy and perplexity of m_2 is $w_1, \dots, w_N$. The cross-entropy of m_2 is then:

$$H_{m_2} = -\frac{1}{N} \log_2 P_{m_2}(w_1^N) = -\frac{1}{N} \log_2 [P_{m_2}(w_1|h_1) \cdots P_{m_2}(w_N|h_N)]$$

where h_i denotes the history at position i . Since m_2 assigns (when predicting the next word) a uniform probability distribution to all the words of its vocabulary V_{m_2} and $|V_{m_2}| = r$, m_2 assigns the same probability $\frac{1}{r}$ to every $w \in V_{m_2}$, at every position i of the test corpus. Assuming that all the words of the test corpus $w_1, \dots, w_N$ are included in V_{m_2} , the probability that m_2 assigns to the correct word w_i at every position i of the test corpus is also $P_{m_2}(w_i|h_i) = \frac{1}{r}$. Therefore, the cross-entropy of m_2 is:

$$H_{m_2} = -\frac{1}{N} \log_2 \left[\frac{1}{r} \cdots \frac{1}{r} \right] = -\frac{1}{N} [\log_2 \frac{1}{r} + \cdots + \log_2 \frac{1}{r}] = -\frac{1}{N} [-N \log_2 r] = \log_2 r$$

and its perplexity is:

$$2^{\log_2 r} = r$$

which is also the assumed perplexity of m_1 . The cross-entropy of m_1 is linked to its perplexity (assumed to be r) by the following formula:

$$2^{H_{m_1}} = r$$

Therefore:

$$\log_2 2^{H_{m_1}} = \log_2 r$$

and the cross-entropy of the two models is also equal:

$$H_{m_1} = \log_2 r = H_{m_2}$$

→ 3. (i) Implement in Python a bigram and a trigram language model for sentences tokenized by a BPE tokenizer. Use add- α smoothing and temperature sampling (no need to use top- k or top- p). Assume that each sentence starts with the pseudo-token **start** (or two pseudo-tokens **start1**, **start2** for the trigram model) and ends with the pseudo-token **end**. Train your models on a training subset of a corpus (e.g., a subset of a corpus included in NLTK – see <http://www.nltk.org/>). You may use an existing pre-trained sub-word tokenizer, or train an existing sub-word tokenizer on the training subset, or implement your own sub-word tokenizer and train it on the training subset (see, e.g., <https://huggingface.co/learn/llm-course/chapter6/1>). Use the same vocabulary in the bigram and trigram models.

Use a separate development subset of the corpus (the same for both models) to tune any hyper-parameters (e.g., α and vocabulary size). You only need to try a few indicative values of the hyper-parameters. This is just an exercise; there is no need to spend too much time on hyper-parameter tuning. Similarly, you can reduce the sizes of the training, development, test subset to allow your experiments to run in reasonable time; we are not trying to create a state-

of-the-art n -gram model, the point of the exercise is to show that you can use in practice the concepts of the corresponding lectures.

(ii) Estimate the cross-entropy and perplexity of your two models on a test subset of the corpus (the same for both models), treating the entire test subset as a single sequence of sentences, with **start** (or **start1**, **start2**) at the beginning of each sentence, and **end** at the end of each sentence. Do not include probabilities of the form $P(*start*|...)$ or $P(*start1*|...)$, $P(*start2*|...)$ in the computation of cross-entropy and perplexity, since we are not predicting the start pseudo-tokens; but include probabilities of the form $P(*end*|...)$, since we do want to be able to predict if a word will be the last one of a sentence. You must also count **end** tokens (but not **start**, **start1**, **start2** tokens) in the total length N of the test subset.

(iii) Show (using appropriate figures and/or tables) the effect of the size of the training subset on the cross-entropy and perplexity of your models, by repeating the training, hyper-parameter tuning, and testing of your models with a few different sizes of the training subset (e.g., 25%, 50%, 75%, 100% of the training data). Keep the development and test subsets fixed (and the same for both models) in all the repetitions.

(iv) Write code to show how your bigram and trigram language models can auto-complete an incomplete sentence. For example, given “I would like to commend the” (slide 45), generate completions like “rapporteur on his work **end**” or “president of the Commission on his intervention **end**”. You need to re-assemble generated sub-words into proper words. Demonstrate (with a few examples of generated sentences) the effect of modifying the temperature (without repeating the training and hyper-parameter tuning for different temperature values). Confirm (showing a few examples of generated texts) that your trigram model generates more fluent texts than your bigram model. Show also some interesting cases (e.g., good and bad sentence completions) when using your two models.

(v) **[Optional, 10% extra bonus]:** Study the paper “Scalable watermarking for identifying large language model outputs” by Dathathri et al. (Nature, October 2024, <https://www.nature.com/articles/s41586-024-08025-4>) and add the watermarking mechanism it describes (its simplest form) to the temperature sampling decoders of your two n -gram language models. Implement the corresponding scoring function (watermark detector) too. Use different watermarking keys for your bigram and trigram model. Show (with examples) if/when the watermarking can distinguish texts generated by the two models. You may drastically reduce the size of the vocabulary and use different temperatures, compared to the ones you used in previous sub-questions.

You are allowed to use existing tools and libraries (e.g., NLTK, Hugging Face) for sentence splitting, sub-word tokenization, counting n -grams, softmax, sampling from distributions, temperature sampling, hashing, but you should write your own code (without collaborating with others) for everything else (e.g., estimating probabilities, computing cross-entropy and perplexity). You may want to compare, however, the cross-entropy and perplexity results of your implementation to results from existing tools and libraries.

Do not forget to include in your report:

- A short description of the algorithms/methods that you used, including a discussion of how you tuned any hyper-parameters, and how you prepared your data (e.g., tokenization and other preprocessing steps, training/development/test splits).
- Cross-entropy and perplexity scores for each model (bigram, trigram) for sub-question (ii), for different sizes of the training subset in sub-question (iii).
- Input/output examples demonstrating how your sentence completion works with the two language models, their fluency, the effect of temperature, interesting cases for sub-question (iv).

- A short description of the watermarking mechanisms you implemented for sub-question (v), if you answered it, and examples demonstrating if/when the watermarking can distinguish texts generated by the two models.

4. (a) Using the following sentences as a tiny training corpus:

<start> he plays football
 <start> he plays cricket
 <start> she enjoys good football
 <start> she plays good music
 <start> he prays to god
 <start> please buy me the other ball
 <start> he pleases the other players by playing good football

estimate the probabilities $P(t_1^4)$ that a bigram model with Laplace smoothing would return, for each one of the following sentences t_1^4 :

t_1^4 : <start> he please god football
 t_1^4 : <start> he plays good football

Assume that the vocabulary V contains all the words of the training corpus (excluding <start>), hence $|V| = 21$. You do not need to perform numeric calculations that can easily be computed with a calculator.

Answer: The bigram language model estimates $P(<start>, \text{he, please, god, football})$ as:

$$P(\text{he} | <start>) P(\text{please} | \text{he}) P(\text{god} | \text{please}) P(\text{football} | \text{god}) = \\ (4+1)/(7+21) \quad (0+1)/(4+21) \quad (0+1)/(1+21) \quad (0+1)/(1+21)$$

where we used Laplace smoothing.

Similarly for $P(<start>, \text{he, plays, good, football})$. (Write down the calculations by yourself.)

Note: In practice, we avoid multiplying probabilities, because they often lead to very small numbers that cannot be represented well by computers. Instead, we usually compute the logarithm of the probability of a word sequence, i.e., we would compute $\log P(<start>, \text{he, please, god, football})$, instead of $P(<start>, \text{he, please, god, football})$, which would lead to the following sum of four logarithms, instead of a product of four probabilities.

$$\log[(4+1)/(7+21)] + \log[(0+1)/(4+21)] + \log[(0+1)/(1+21)] + \log[(0+1)/(1+21)]$$

(b) Assume that a user wrote the sequence w_1^4 using the keyboard of a mobile phone:

w_1^4 : <start> he pls gd ftball

Estimate the probabilities $P(t_1^4 | w_1^k)$ of the two hypotheses (word sequences the user might have intended to write) t_1^4 of sub-question (a), using a noisy channel model (see slides) and the language model of sub-question (a). Assume that $P(w_i | t_i) \cong \frac{1}{LD(w_i, t_i) + 1}$, where $LD(w_i, t_i)$ is Levenshtein edit distance from word w_i to t_i . Show your calculations in detail, without computing the Levenshtein distances.

Answer: Using the noisy channel of the slides:

$$P(t_1^4|w_1^4) = P(t_1^4) P(w_1^4|t_1^4) / P(w_1^4)$$

For $t_1^4 = \text{<start> he please god football}$:

$P(\text{<start>, he, please, god, football})$

$$P(\text{<start> he pls gd ftball} | \text{<start>, he, please, god, football}) / P(w_1^4)$$

The probability $P(\text{<start>, he, please, god, football})$ is estimated by the language model as in sub-question (a).

Assuming $P(w_i|t_i) \cong \frac{1}{LD(w_i,t_i)+1}$, the probability $P(\text{<start> he pls gd ftball} | \text{<start>, he, please, god, football})$ becomes:

$$P(\text{he, he}) P(\text{pls, please}) P(\text{gd, god}) P(\text{ftball, football}) =$$

$$1/(LD(\text{he, he})+1) \ 1/(LD(\text{pls, please})+1) \ 1/(LD(\text{gd, god})+1) \ 1/(LD(\text{ftball, football})+1)$$

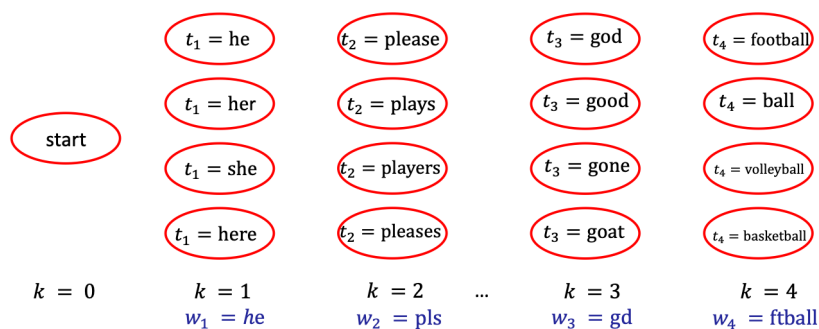
We don't need to compute $P(w_1^4)$, because it is the same for both hypotheses $t_1^4 = \text{<start> he please god football}$ and $t_1^4 = \text{<start> he plays good football}$.

Similarly, we estimate $P(t_1^4|w_1^4)$ for the hypothesis $t_1^4 = \text{<start> he plays good football}$. (Write down the calculations by yourself.) Eventually, we select the hypothesis t_1^4 with the highest $P(t_1^4|w_1^4)$.

Note: In practice, we would again compute the logarithm of each product of probabilities, obtaining a sum of logarithms of probabilities, instead of a product of probabilities.

(c) Explain how the beam search decoder of the slides would work in the following case. We use the noisy channel model of the previous sub-question, with the bigram language model of sub-question (a), trained on the training corpus of that sub-question. The beam size $b = 2$.

Beam search decoder



For $k = 1$, the candidate paths t_1^k are the following. For each path, we compute $P(t_1^k)P(w_1^k|t_1^k)$. (In practice, we would compute the logarithm.) The $b = 2$ best paths are marked with asterisks.

$$\text{<start, he>: } P(\text{he}|\text{start}) P(\text{he}|\text{he}) = (4+1)/(7+21) \ 1/(0+1) = 5/28 = 0.179 \ **$$

$$\text{<start, her>: } P(\text{her}|\text{start}) P(\text{her}|\text{her}) = (0+1)/(7+21) \ 1/(1+1) = 1/28 \ 1/2 = 0.018$$

$$\text{<start, she>: } P(\text{she}|\text{start}) P(\text{he}|\text{she}) = (2+1)/(7+21) \ 1/(1+1) = 3/28 \ 1/2 = 0.054 \ **$$

$$\langle \text{start, here} \rangle: P(\text{here}|\text{start}) P(\text{he}|\text{here}) = (0+1)/(7+21) \cdot 1/(2+1) = 1/28 \cdot 1/3 = 0.012$$

For $k = 2$, the candidate paths t_1^k are the following. Again, we compute $P(t_1^k)P(w_1^k|t_1^k)$ for each path. Complete the calculations by yourselves. Mark the $b = 2$ best paths with asterisks. Notice that we can reuse computations (parts of the products) from $k = 1$.

$\langle \text{start, he, please} \rangle: P(\text{he}|\text{start}) P(\text{he}|\text{he}) P(\text{please}|\text{he}) P(\text{pls}|\text{please}) = \dots$
 $\langle \text{start, he, plays} \rangle: P(\text{he}|\text{start}) P(\text{he}|\text{he}) P(\text{plays}|\text{he}) P(\text{pls}|\text{plays}) = \dots$
 $\langle \text{start, he, players} \rangle: P(\text{he}|\text{start}) P(\text{he}|\text{he}) P(\text{players}|\text{he}) P(\text{pls}|\text{players}) = \dots$
 $\langle \text{start, he, pleases} \rangle: P(\text{he}|\text{start}) P(\text{he}|\text{he}) P(\text{pleases}|\text{he}) P(\text{pls}|\text{pleases}) = \dots$
 $\langle \text{start, she, please} \rangle: P(\text{she}|\text{start}) P(\text{he}|\text{she}) P(\text{please}|\text{she}) P(\text{pls}|\text{please}) = \dots$
 $\langle \text{start, she, plays} \rangle: P(\text{she}|\text{start}) P(\text{he}|\text{she}) P(\text{plays}|\text{she}) P(\text{pls}|\text{plays}) = \dots$
 $\langle \text{start, she, players} \rangle: P(\text{she}|\text{start}) P(\text{he}|\text{she}) P(\text{players}|\text{she}) P(\text{pls}|\text{players}) = \dots$
 $\langle \text{start, she, pleases} \rangle: P(\text{she}|\text{start}) P(\text{he}|\text{she}) P(\text{pleases}|\text{she}) P(\text{pls}|\text{pleases}) = \dots$

For $k = 3$, the candidate paths t_1^k are the following. Again, we compute $P(t_1^k)P(w_1^k|t_1^k)$ for each path. Complete by yourselves. Mark the $b = 2$ best paths with asterisks

...

For $k = 4$, the candidate paths t_1^k are the following. Again, we compute $P(t_1^k)P(w_1^k|t_1^k)$ for each path. Complete by yourselves. Which hypothesis (path) $\langle \text{start}, t_1, t_2, t_3, t_4 \rangle$ is selected?

...