ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Large Language Models
Spring 2024

Prof. Ryan Cotterell

Assignment 1

13/03/2024 - 15:17h

Disclaimer: In full transparency, some of the questions on this assignment are modified versions of recent NLP publications that treat language models formally. The relevant publications are not difficult to find. Our choice to base the assignment questions on recent results should assure you that the questions introduce you to recent research in the field and that the questions are interesting—indeed, they are *so* interesting active researchers succeeded in publishing their answers in well-respected venues! In principle, you *can* simply copy the solutions to the questions below from the original text in some cases. Moreover, unless such copying is done sloppily,[1] i.e., if you do not take care to cover your tracks, you will likely get away with it. However, we strongly advise against such a tactic. First, while it may sound hackneyed, you are simply cheating yourself. A classroom is a controlled environment where you can learn under the guidance of other humans who understand the subject matter. You ought to take advantage of this opportunity. Indeed, that is presumably why you came to ETHZ in the first place. More practically, the nature of the questions on the assignments is meant to, to a large extent, resemble the possible questions in the final exam and are meant to prepare you for it. Any content covered here is fair game for the exam. Therefore, we strongly recommend that you *think* about the problems *yourself* and try to solve them on your own. People learn mathematics best by doing it.

The **deadline** for submitting the first assignment is **Tuesday, April 30th, at 23:59**.

You can obtain at most 100 points in this assignment. The grading scheme is the following:

Final assignment grade	Total number of points
6.0	$[96, 100]$
5.75	$[92, 96)$
5.5	$[86, 92)$
5.25	$[82, 86)$
5.0	$[72, 82)$
4.75	$[64, 72)$
4.5	$[58, 64)$
4.25	$[50, 58)$
4.0	$[40, 50)$

[1] If we can prove you copied, we will report you to the rectorate. Beware!

Question 1: Measure over Trees (24 pts)

The purpose of this question is to get you to explore measure-theoretic probability more deeply. To that end, we are going to ask you to construct a probability space for a probabilistic context-free grammar (PCFG). Consider the following context-free grammar (CFG)

$$\begin{aligned} \mathrm{S} &\rightarrow \mathrm{S}\,\mathrm{S} \\ \mathrm{S} &\rightarrow a \end{aligned} \tag{1}$$

where S is the only nonterminal and a is the only terminal.

a) **(8 pts)** Determine an appropriate sample space Ω for trees generated by this CFG. Use a construction similar to the cylinder algebra as shown in lecture notes and construct an algebra over Ω. **Hint**: Remember to take into account that a tree can be infinite.

Now, suppose we put probabilities on the derivations to make Eq. (1) a PCFG as follows:

$$p\left(\mathrm{S} \rightarrow \mathrm{S}\,\mathrm{S}\right) = q \tag{2}$$

$$p\left(\mathrm{S} \rightarrow a\right) = 1 - q \tag{3}$$

where $0 < q < 1$.

b) **(8 pts)** Based on Eq. (2), construct a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ over the previous measurable space by first constructing a pre-measure for your algebra in part (a). Remember to prove that your construction satisfies the conditions of a probability space.

c) **(8 pts)** Next, provide an appropriate characterization of tightness for the PCFG in Eq. (2). Next, determine and *prove* the condition on q such that the PCFG in Eq. (2) is tight.

Hint: While it would be educational to derive this from first principles yourself, there is a general case given in the course notes that can be specialized if you are stuck.

Question 2: The Moments of String Length (24 pts)

In this question, we explore one of the benefits of having a language model based on a simpler formalism—namely, many computations that we may wish to perform can be done *exactly* in polynomial time. In that context, your task is to derive efficient algorithms for computing several quantities about the string length under two modeling frameworks

i) when the language model is *finite-state* and

ii) when the language model is *context-free*.

In this question, the following two quantities are of interest

1. The expected string length μ:
$$\mu \stackrel{\text{def}}{=} \sum_{\boldsymbol{y} \in \Sigma^*} p_{\text{LM}}(\boldsymbol{y}) |\boldsymbol{y}| \tag{4}$$

2. The expected square deviation of the string length from the mean σ^2:
$$\sigma^2 \stackrel{\text{def}}{=} \sum_{\boldsymbol{y} \in \Sigma^*} p_{\text{LM}}(\boldsymbol{y}) \left(|\boldsymbol{y}| - \mu\right)^2 \tag{5}$$

Let $\mathcal{A} = (\Sigma, Q, \delta, \lambda, \rho)$ be a weighted finite-state automaton with no ε-transitions encoding a finite-state language model p_{LM}.

a) **(10 pts)** Give an algorithm for computing the mean string length (cf. Eq. (4)) when p_{LM} is a finite-state language model. The algorithm should run in time $\mathcal{O}\left(|Q|^3\right)$. Show that your algorithm indeed runs in time $\mathcal{O}\left(|Q|^3\right)$.

Hint: This problem requires a very simple modification to the allsum algorithm, which is discussed in the notes; indeed, if you set up the problem right, you can call the allsum algorithm as a black box.

b) **(10 pts)** Give an algorithm for computing the variance of the string length (cf. Eq. (5)) when p_{LM} is a finite-state language model. The algorithm should run in time $\mathcal{O}\left(|Q|^3\right)$. Show that your algorithm indeed runs in time $\mathcal{O}\left(|Q|^3\right)$.

Hint: You may wish to introduce a form of mean-centering by introducing negative lengths for certain symbols.

Let $\mathcal{G} = (\Sigma, \mathcal{N}, \mathrm{S}, \mathcal{P}, \mathcal{W})$ be a weighted context-free grammar encoding a context-free language model p_{LM}.

c) **(4 pts)** Give a simple fixed-point procedure for computing the mean string length (cf. Eq. (4)) under a context-free language model p_{LM}. In contrast to the previous two approaches, this problem requires an iterative algorithm. Analyze the algorithms's space and time complexity.

Question 3: Efficiently Simulating Bounded Stacks with RNNs (20 pts)

As we discussed in the lecture, *hierarchical structure*, characterized by long distance and nested dependencies, lies at the core of the human language. Indeed, this motivated our theoretical discussion of context-free grammars as a useful paradigm for describing language. Studying modern language models with respect to their ability to efficiently represent hierarchical structure, therefore, provides evidence that they are useful models for human language. This question directly addresses this point. More precisely, we will study the ability of recurrent neural network language models to recognize a variant of the $\mathrm{D}(k)$ languages.

As introduced in the lecture notes, the $\mathrm{D}(k)$ languages are in a way archetypal context-free languages. Recognizing $\mathrm{D}(k)$ languages is conceptually simple—a system has to *remember* the sequence of currently non-closed opening brackets and make sure they are closed in the correct order, at which point the closed bracket pairs can be "forgotten", i.e., popped off the stack. This means that the memory necessary to recognize any string in $\mathrm{D}(k)$ is proportional to the number of *non-closed brackets* at any time. We formalize it by counting how many more open brackets than closed brackets there are at each timestep in the string:

$$d(\boldsymbol{y}_{\leq t}) \stackrel{\text{def}}{=} \mathsf{count}(\boldsymbol{y}_{\leq t}, \langle) - \mathsf{count}(\boldsymbol{y}_{\leq t}, \rangle) \tag{6}$$

where $\mathsf{count}(\boldsymbol{y}_{\leq t}, \langle)$ refers to the number of times *any* opening bracket occurs in $\boldsymbol{y}_{\leq t}$ and $\mathsf{count}(\boldsymbol{y}_{\leq t}, \rangle)$ the number of times *any* closing bracket occurs in $\boldsymbol{y}_{\leq t}$.

While context-free languages like $\mathrm{D}(k)$ describe arbitrarily deep hierarchical structures, natural languages exhibit bounded nesting in practice, as discussed in the lecture. Furthermore, the infinite nesting and therefore infinitely long stacks also make it impossible to represent context-free languages with finite precision. In this question, we investigate how to represent $\mathrm{D}(k)$ languages which can only nest up to some *bounded* depth m. We denote such languages as $\mathrm{D}(k, m)$.

Definition 1 ($\mathrm{D}(k, m)$ languages)**.** *Let $k, m \in \mathbb{N}$. We define the* ***bounded Dyck language*** $\mathrm{D}(k, m)$ *by combining* $\mathrm{D}(k)$ *with a bound on the nesting depth:*

$$\mathrm{D}(k, m) \stackrel{\text{def}}{=} \{\boldsymbol{y} \in \mathrm{D}(k) \mid d(\boldsymbol{y}_{\leq t}) \leq m, t = 1, \ldots, T\}, \tag{7}$$

where T corresponds to the length of the string.

Due to their bounded nesting depth, $\mathrm{D}(k, m)$ languages can be recognized by stacks of bounded depth and therefore with *bounded memory*—this means that they are in fact *finite-state*. This makes them especially well-suited as a benchmark for finite-precision language models.

This question is roughly divided into two parts: in the first part, you will show that an Elman RNN is able to simulate a finite-state automaton that recognizes the $\mathrm{D}(k, m)$ for some k and m. In the second part of the question, you are asked to show that the RNN indeed *recognizes* the language as well using a specific definition of acceptance.

We begin with a warm-up question.

a) **(1 pt)** Suppose that the current bounded stack configuration in an automaton recognizing $L = \mathrm{D}(2, 3)$ is

$$\begin{aligned}\boldsymbol{\gamma}_1 &= \langle_2 \\ \boldsymbol{\gamma}_2 &= \langle_1 \langle_1 \langle_1 \\ \boldsymbol{\gamma}_3 &= \langle_2 \langle_1 \langle_2\end{aligned}$$

What are the new stack configurations $\boldsymbol{\gamma}_1'$, $\boldsymbol{\gamma}_2'$, $\boldsymbol{\gamma}_3'$ after reading in each of the following symbols (each one starting from a stack $\boldsymbol{\gamma}_1$, $\boldsymbol{\gamma}_2$, $\boldsymbol{\gamma}_3$, *not* one after another)?

1. $\rangle_2$
2. $\rangle_1$
3. $\langle_2$

Use $\emptyset$ to denote an empty stack and simply state that the automaton would reject a string if the processed string is not in $\mathrm{D}(2,3)$.

Note: You have to specify *nine* stack configurations altogether.

We next prove that $\mathrm{D}(k,m)$ languages are in fact finite-state by constructing an FSA recognizing $\mathrm{D}(k,m)$.

b) **(4 pts)** Let $k, m \in \mathbb{N}$. Construct a finite-state automaton

$$\mathcal{A}_{\mathrm{D}(k,m)} = \left(\Sigma_{\mathcal{A}_{\mathrm{D}(k,m)}}, Q_{\mathcal{A}_{\mathrm{D}(k,m)}}, \delta_{\mathcal{A}_{\mathrm{D}(k,m)}}, I_{\mathcal{A}_{\mathrm{D}(k,m)}}, F_{\mathcal{A}_{\mathrm{D}(k,m)}}\right) \tag{8}$$

by defining the elements of the tuple such that $\mathcal{A}_{\mathrm{D}(k,m)}$ accepts the $\mathrm{D}(k,m)$, i.e. $L\left(\mathcal{A}_{\mathrm{D}(k,m)}\right) = \mathrm{D}(k,m)$. Note that we are only interested in binary recognition of the language.

Prove that your constructed automaton really recognizes exactly the required languages and give an big-$\mathcal{O}$ bound on the number of states and transitions in $\mathcal{A}_{\mathrm{D}(k,m)}$ in terms of k and m

Hint: The automaton should encode each possible configuration of the bounded stack as a separate state and define the outgoing transitions based on what continuations of the string encoded by the stack are possible. For example, in $\mathrm{D}(2,2)$, the only "valid" accessible state from $[\langle_1 \langle_2]$ would be $[\langle_1]$ after reading the input symbol $\rangle_2$. Use the notation $[\boldsymbol{y}]$ to denote the state corresponding to the stack sequence $\boldsymbol{y}$. To make reasoning later easier, we suggest you use a designated "accept" state (the only final state) A and a "rejecting state" R. For example, the input symbol $\langle_2$ would lead from $[\langle_1 \langle_2]$ to R since the stack bound would otherwise be exceeded.

All the elements of the tuple have to be formally defined, but you can reason about your decisions more informally.

Intuitively, a stack bounded to m symbols can encode up k^m different configurations (this many different *states*). Recall that, to represent such a language with an Elman RNN through Minsky's construction directly, we would require a hidden state of size $\mathcal{O}\left(k^m\right)$—this is untractable even for small k and m. However, besides being finite-state, the $\mathrm{D}(k,m)$ languages have a very clear and useful structure which you hopefully already exploited when defining the finite-state automata accepting them. As we will see, this structure also makes them efficiently representable using an RNN.

You will now be guided through a construction of an Elman RNN that simulates a bounded stack and is thus able to recognize the $\mathrm{D}(k,m)$ language. Assuming that the automaton $\mathcal{A}_{\mathrm{D}(k,m)}$ encodes the $\mathrm{D}(k,m)$ language correctly, we will try to represent this machine with an Elman RNN. To do so, we will, similarly to the Minsky construction, describe how the RNN hidden states represent the FSA states and how the recurrence relation of the RNN encodes the transition function of the $\mathcal{A}_{\mathrm{D}(k,m)}$. More precisely, you will construct an RNN with the hidden state size of $\mathcal{O}\left(mk\right)$—a vast improvement on the one based on Minsky's construction.

As mentioned, when processing strings in $\mathrm{D}(k,m)$, we only have to keep the opening brackets on the stack as the summary of the entire string so far, $\boldsymbol{y}_{<t}$, to be able to determine which next symbols are allowed—this gives us a complete characterization of the string and how it can be continued. We, therefore, choose to encode a *string* $\boldsymbol{y}_{<t}$ by encoding the *stack* it induces in the hidden state (its *context encoding* in the vocabulary from the notes) of the RNN summarizing $\boldsymbol{y}_{<t}$ as follows. We define the column vector

$$\mathrm{enc}\left(\left[\langle_{i_1} \langle_{i_2} \cdots \langle_{i_{m'}}\right]\right) \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{e}_{i_{m'}} \\ \mathbf{e}_{i_{m'-1}} \\ \vdots \\ \mathbf{e}_{i_2} \\ \mathbf{e}_{i_1} \\ \mathbf{0}_{\in(m-m')k} \end{pmatrix} \in \mathbb{R}^{k \cdot m}, \tag{9}$$

where $\mathbf{e}_n$ represents the n^{th} canonical basis vector. The indices $i_j \in \{1, \ldots, k\}$ refer to the identities of the brackets in $\Sigma_{\mathcal{A}_{\mathrm{D}(k,m)}}$ and m' refers to the current depth of the stack. The element encoded by the "top" $(i_{m'})$ of the hidden state, $\mathbf{e}_{i_{m'}}$, represents the symbol on the top of the stack.

c) **(1 pt)** What are the embedings enc $(\boldsymbol{\gamma}_i)$ for $i = 1, 2, 3$ from question *(a)*? What are the embeddings of the new stacks after reading in the same new inputs as in question *(a)*?

We now start building the RNN recognizing the $\mathrm{D}(k, m)$ language.

d) **(4 pts)** Before we define the Elman RNN recognizing $\mathrm{D}(k, m)$, let us first consider a more flexible "RNN" model *without* an activation function, in which the recurrence matrix $\mathbf{U}$ *depends* on the current input. We model the dynamics map as

$$\mathbf{h}_t = \mathbf{U}(y_t)\,\mathbf{h}_{t-1} + \mathbf{V}\mathbf{o}_{y_t}, \tag{10}$$

where $\mathbf{U} : \overline{\Sigma} \to \mathbb{R}^{k\cdot m \times k\cdot m}$ is a *function* of the current input y_t and $\mathbf{V} \in \mathbb{R}^{k\cdot m\times(2\cdot k+1)}$. $\mathbf{o}_{y_t}$ refers to the one-hot-encoding of the symbol y_t. Given the encoding from Eq. (9), define the recurrence matrix function $\mathbf{U}(\cdot)$ and the input matrix $\mathbf{V}$ such that, assuming that $\mathbf{h}_{t-1}$ encodes some state $q \in Q_{\mathcal{A}_{\mathrm{D}(k,m)}}$ (and thus the stack at time step $t-1$), $\mathbf{h}_t$ encodes the state $q' \in Q_{\mathcal{A}_{\mathrm{D}(k,m)}}$ such that $\mathcal{A}_{\mathrm{D}(k,m)}$ transitions into q' from q upon reading y_t (and thus $\mathbf{h}_t$ encodes the state of the stack at time t). Show that the dynamics map you constructed works as desired by showing how the transitions in the hidden state space of the RNN correspond to the transitions in the automaton $\mathcal{A}_{\mathrm{D}(k,m)}$.

Hint: This question should help you build the intuition of how the stack represented using Eq. (9) can be manipulated (pushed onto and popped from) using matrix multiplications. $\mathbf{U}(y)$ should be a simple function of the input symbol depending only on whether the stack should be *popped from* or *pushed onto*. You will then reuse the different possible values of $\mathbf{U}(\cdot)$ when constructing an actual Elman RNN.

e) **(5 pts)** Now, suppose that we have a standard Elman RNN, in which the recurrence matrix is *fixed*:

$$\mathbf{h}_t = H\left(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{V}\mathbf{o}_{y_t} + \mathbf{b}\right), \tag{11}$$

where H refers to the Heaviside activation function as defined in the lecture. Define the parameters $\mathbf{U} \in \mathbb{R}^{2\cdot k\cdot m\times 2\cdot k\cdot m}$, $\mathbf{V} \in \mathbb{R}^{2\cdot k\cdot m\times(2\cdot k+1)}$ and $\mathbf{b} \in \mathbb{R}^{2\cdot k\cdot m}$ such that the update steps of the RNN again correspond to the transitions in the automaton $\mathcal{A}_{\mathrm{D}(k,m)}$ as in the previous subquestion.

Show that the dynamics map you constructed works as desired by showing how the transitions in the hidden state space of the RNN correspond to the transitions in the automaton $\mathcal{A}_{\mathrm{D}(k,m)}$.

Hint: Since the recurrence matrix $\mathbf{U}$ cannot depend on whether a symbol should be pushed or popped anymore, consider performing *both* operations at once. For this, you should "duplicate" the embeddings of the stack into the RNN hidden state (as hinted by the matrix dimensions). Then, both the popping and pushing operations can be performed—one on each half of the hidden state—and, upon performing both operations using the recurrence matrix, the correct half can be "selected" using the "embedding" $\mathbf{V}\mathbf{o}_{y_t}$ of the input symbol y_t and the activation function (by "selected", we mean that the non-relevant half of the embedding should be zeroed-out after adding the embedding of the input symbol and passing the sum through the activation function).

Phew! We now have an Elman RNN whose hidden state updates correspond to the transitions of the automaton $\mathcal{A}_{\mathrm{D}(k,m)}$ recognizing $\mathrm{D}(k, m)$—this alone is a lot already! It means that the RNN captures the "dynamics" of the language. We now just have to figure out a way to make it "recognize" it formally. In contrast to the formal treatment of weighted languages in the lectures, we will be studying the *binary* acceptance of *unweighted* languages. For that, we must define what it means for an RNN (defining string or next symbol probabilities, i.e., a weighted language) to generate an *unweighted* language, such as $\mathrm{D}(k, m)$. The key idea is to define recognition in terms of the *local* next symbol probabilities $p_{\mathrm{SM}}(y_t \mid \boldsymbol{y}_{<t})$ instead of the full string probabilities $p_{\mathrm{LN}}(\boldsymbol{y})$—those must approach zero with sequence length.

Definition 2 (Locally η-truncated support)**.** *Let $\eta \in [0, 1]$. The **locally η-truncated support** of a sequence model p_{SM} is the set*

$$L_\eta(p_{SM}) \stackrel{\text{def}}{=} \{\boldsymbol{y} \in \Sigma^* \colon p_{SM}(y_t \mid \boldsymbol{y}_{<t}) \geq \eta, t = 1 \ldots |\boldsymbol{y}|\}.$$

This is the set of strings for which the model assigns at least η probability to each symbol conditioned on the string before it. Intuitively, these are the strings for which the model does not strongly "disagree" with the next symbol (assign it probability $< \eta$) at any point.

f) **(5 pts)** Given the stack encoding from Eq. (9) and one-hot encodings $\mathbf{o}_y$ of the input symbols construct the output matrix $\mathbf{E}$ and the bias vector $\mathbf{b}'$ such that

$$L_\eta\left(p_{\text{SM}}\right) = \mathrm{D}(k, m) \tag{12}$$

where the p_{SM} is the sequence model whose conditional probabilities are defined by the constructed RNN, i.e.,

$$p_{\text{SM}}\left(y_t \mid \boldsymbol{y}_{<t}\right) = \operatorname{softmax}\left(\mathbf{E}\,\mathbf{h}_{t-1} + \mathbf{b}'\right)_{y_t} \tag{13}$$

Notice that we use an additional bias term $\mathbf{b}'$ in the constructed probabilities compared to the formulation from the lecture notes. While this does not increase the expressivity of the model, it makes the definition of the parameters easier.

Hint: To satisfy the definition of η-truncated support, you should show that $p_{\text{SM}}\left(y_t \mid \boldsymbol{y}_{<t}\right) \geq \eta$ if and only if the symbol y_t is an "acceptable" continuation of the input string $\boldsymbol{y}_{<t}$. For example, since $\langle_1$ and $\rangle_1$ are valid continuations of the string $\langle_1\langle_2\rangle_2\langle_1$ in $\mathrm{D}(2,3)$, it should hold that $p_{\text{SM}}\left(\langle_1 \mid \langle_1\langle_2\rangle_2\langle_1\right) \geq \eta$ and $p_{\text{SM}}\left(\rangle_1 \mid \langle_1\langle_2\rangle_2\langle_1\right) \geq \eta$. On the other hand, since $\langle_2$ and $\rangle_1$ are not valid continuations of the string $\langle_2\langle_1\langle_1\rangle_1\langle_2$, it should hold that $p_{\text{SM}}\left(\langle_2 \mid \langle_2\langle_1\langle_1\rangle_1\langle_2\right) < \eta$ and $p_{\text{SM}}\left(\rangle_1 \mid \langle_2\langle_1\langle_1\rangle_1\langle_2\right) < \eta$. To do so, introduce a real-valued parameter α into your definition of the embedding matrix which will allow you to control the nature of the conditional distributions defined by your constructed RNN. Then, determine the (bounds) on the parameter α by computing the different possible normalization constants (there will be three relevant ones) and ensuring that the conditions on the conditional probabilities are satisfied by setting or bounding α appropriately.

Question 4: Tightness of Trained RNN Language Models (10 pts)

In this question, we investigate the tightness of RNN language models based on the Elman RNN. We will construct a non-tight RNN language model step-by-step to get a qualitative understanding of the requirements for RNN language models to be tight and some scenarios when they can be non-tight.

First, recall the definition of the Elman RNN recurrence with the ReLU activation function and the one-hot encoding function $[\![\cdot]\!]$:

$$\mathbf{h}_t \stackrel{\text{def}}{=} \text{ReLU}(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{V}[\![y_t]\!] + \mathbf{b}_h) \tag{14}$$

where, in general, $\mathbf{U} \in \mathbb{R}^{D\times D}, \mathbf{V} \in \mathbb{R}^{D\times|\Sigma|}, \mathbf{b}_h \in \mathbb{R}^D$. Given an Elman RNN $\mathcal{R}$, the distribution of the next symbol is defined as

$$p_{\text{SM}}(y_t \mid \boldsymbol{y}_{<t}) \stackrel{\text{def}}{=} \text{softmax}(\mathbf{E}\mathbf{h}_{t-1})_{y_t} \tag{15}$$

where $y_t \in \Sigma$. The RNN is **rational** if $\mathbf{U} \in \mathbb{Q}^{D\times D}, \mathbf{V} \in \mathbb{Q}^{D\times|\Sigma|}, \mathbf{b}_h \in \mathbb{Q}^D$.

a) **(1 pts)** Show that for any input string $\boldsymbol{y} \in \Sigma^*$, the hidden state $\mathbf{h}_t$ of a rationally-weighed RNN is always rational, that is, $\mathbf{h}_t \in \mathbb{Q}^D$ for all $t \in \{1, 2, \cdots, |\boldsymbol{y}|\}$.

Now, let us focus on the tightness of *single-symbol* rational-weighted RNNs, in which $\Sigma = \{a\}$. The probability of a^T given by $\mathcal{R}$ can be written as

$$p_{\text{LN}}(a^T) = \begin{cases} p_{\text{SM}}(\text{EOS} \mid a^T) \prod_{t=0}^{T-1} p_{\text{SM}}(a \mid a^t) & \textbf{if } T > 0 \\ p_{\text{SM}}(\text{EOS} \mid a^0) & \textbf{otherwise} \end{cases}. \tag{16}$$

$\mathcal{R}$ is said to be **tight** if

$$Z = \sum_{T=0}^{\infty} p_{\text{LN}}(a^T) = 1, \tag{17}$$

otherwise, $\mathcal{R}$ is said to be **non-tight**.

Attention: In this subquestion, you are *not* supposed to rely on any existing tightness results from the lecture notes, Chen et al. [2018], or Du et al. [2022] as a shortcut to the solution. Please *only* use Eq. (17) and compute Z directly.

b) **(1 pts)** Show that when $p_{\text{SM}}\left(a \mid a^t\right)$ is constant, i.e., $p_{\text{SM}}\left(a \mid a^t\right) \equiv c$ for some $c \in (0, 1)$, $\mathcal{R}$ is tight.

c) **(2 pts)** Show that when $p_{\text{SM}}\left(a \mid a^t\right) = \frac{(t+1)}{1+(t+1)}$, $\mathcal{R}$ is tight.
Hint: You can use that $\frac{1}{(n+1)(n+2)} = \frac{1}{n+1} - \frac{1}{n+2}$.

We have so far exhibited two tight RNN LMs. We now turn to non-tight ones. Intuitively, if the LM keeps putting more weight on continuing the generation of symbols and less weight on terminating with EOS, some generations will never stop. This results in a non-tight LM. We will construct a family of conditional probabilities $p_{\text{SM}}\left(a \mid a^t\right)$ resulting in a non-tight model and then show how they can be implemented by an RNN LM.

d) **(2 pts)** Show that when $p_{\text{SM}}\left(a \mid a^t\right) = \frac{\exp(t)}{1+\exp(t)}$, $\mathcal{R}$ is *non-tight*.
Hint: Compare $p_{\text{LN}}(a^T)$ with the one in the previous question.

Let us begin the construction of a non-tight RNN, which gives us the distribution $p_{\text{SM}}\left(a \mid a^t\right) = \frac{\exp(t)}{1+\exp(t)}$ at every timestep t.

e) **(2 pts)** Design a rational-weighted RNN $\mathcal{R}$ with $D = 1, \mathbf{h}_0 = -1, \mathbf{E} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ such that $p_{\text{SM}}\left(a \mid a^t\right) = \frac{\exp(t-1)}{1+\exp(t-1)}$ for all $t \in \mathbb{N}_{\geq 0}$.[2] Give $\mathbf{U}, \mathbf{V}, \mathbf{b}_h$.

The previous subquestion showed that there exists a setting of RNN weights that results in a non-tight model. A reasonable question is whether such weights can be found during *training* on a dataset using gradient descent. We show that this indeed is the case by showing that an RNN trained until convergence on finite data with gradient descent can be non-tight.

f) **(2 pts)** Consider the dataset $\mathcal{D} = \left\{a^1\right\}$. Given the RNN $\mathcal{R}$ you constructed in the previous subquestion, compute $\frac{\partial p_{\text{LN}}(a^1)}{\partial \mathbf{U}}$, $\frac{\partial p_{\text{LN}}(a^1)}{\partial \mathbf{V}}$, and $\frac{\partial p_{\text{LN}}(a^1)}{\partial \mathbf{b}_h}$. Based on this, conclude why RNNs trained on finite datasets can be non-tight. Here, we define $\left.\frac{\partial \text{ReLU}(x)}{\partial x}\right|_{x=0} = 0$.

[2] Note that the second dimension of the softmax vector corresponds to the EOS symbol.

Question 5: Efficient Transformers (22 pts)

The quadratic time complexity of attention with respect to the length of the input sequences, has posed a significant obstacle for the effective use of transformer-based models on long inputs. In this question we will work on building a kernel-based method for computing attention efficiently.

We start by providing the definition of kernel function.

Definition 3 (Positive definite kernel)**.** *Let $\mathbb{V}$ be a Hilbert space and $\langle \cdot, \cdot \rangle_{\mathbb{V}}$ its corresponding inner product. A **positive definite kernel** $K(\mathbf{x}, \mathbf{y}) : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{V}$ is a function for which there exists a feature map $\phi : \mathbb{R}^D \to \mathbb{V}$ so that:*

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathbb{V}} \tag{18}$$

In words, a kernel is a function that can always be defined as an inner product in a (possibly infinite-dimensional) feature space. A well-known kernel is the **Gaussian Kernel** $K_{\mathrm{G}}(\mathbf{x}, \mathbf{y})$ which has the following form.

$$K_{\mathrm{G}}(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2}\right) \tag{19}$$

a.1) **(2 pts)** Assuming $D = 1$, provide the feature map $\phi_{\mathrm{G}}(\mathbf{x})$ of the Gaussian Kernel so that the following equation holds.

$$K_{\mathrm{G}}(\mathbf{x}, \mathbf{y}) = \langle \phi_{\mathrm{G}}(\mathbf{x}), \phi_{\mathrm{G}}(\mathbf{y}) \rangle \tag{20}$$

We now introduce the **Softmax Kernel**, which is defined as follows.

$$K_{\mathrm{SM}}(\mathbf{x}, \mathbf{y}) \stackrel{\text{def}}{=} \exp\left(\mathbf{x}^\top \mathbf{y}\right). \tag{21}$$

b.1) **(2 pts)** Prove that the Softmax Kernel $K_{\mathrm{SM}}(\mathbf{x}, \mathbf{y})$ is related to the Gaussian Kernel $K_{\mathrm{G}}(\mathbf{x}, \mathbf{y})$ through the following expression.

$$K_{\mathrm{SM}}(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{||\mathbf{x}||^2}{2}\right) K_{\mathrm{G}}(\mathbf{x}, \mathbf{y}) \exp\left(\frac{||\mathbf{y}||^2}{2}\right) \tag{22}$$

b.2) **(1 pts)** Assuming $D = 1$, provide the feature map $\phi_{\mathrm{SM}}(\mathbf{x})$ of the Gaussian Kernel so that the following equation holds.

$$K_{\mathrm{SM}}(\mathbf{x}, \mathbf{y}) = \langle \phi_{\mathrm{SM}}(\mathbf{x}), \phi_{\mathrm{SM}}(\mathbf{y}) \rangle \tag{23}$$

We introduce a theorem that allows us to approximate any stationary kernel (such as the Gaussian or the Softmax kernel) through a randomized feature map $\xi(\cdot)$.

Theorem 1 (Bochner's Theorem)**.** *Let $\xi(\cdot) : \mathbb{R}^D \to \mathbb{R}^r$ be a random feature map defined as follows.*

$$\xi(\mathbf{x}) = \frac{h(\mathbf{x})}{\sqrt{m}} \left[f_1(\boldsymbol{\omega}_1^\top \mathbf{x}), \ldots, f_1(\boldsymbol{\omega}_m^\top \mathbf{x}), \ldots, f_l(\boldsymbol{\omega}_1^\top \mathbf{x}), \ldots, f_l(\boldsymbol{\omega}_m^\top \mathbf{x}) \right] \tag{24}$$

where $r \stackrel{\text{def}}{=} m \cdot l$, $f_1, \ldots, f_l : \mathbb{R} \to \mathbb{R}$, $h : \mathbb{R}^D \to \mathbb{R}$ and $\boldsymbol{\omega}_i \in \mathbb{R}^D$ are drawn independently from $\mathcal{D}$ for a probability distribution $\mathcal{D}$.
A stationary kernel $K(\mathbf{x}, \mathbf{y})$ can be approximated as:

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathbb{V}} = \mathbb{E}_{\boldsymbol{\omega} \sim \mathcal{D}} \left[\langle \xi(\mathbf{x}), \xi(\mathbf{y}) \rangle_{\mathbb{V}} \right] \tag{25}$$

Bochner's theorem can be used to approximate both the Gaussian and the Softmax kernels.

Proposition 1. *The Gaussian Kernel* $K_{\mathrm{G}}(\mathbf{x}, \mathbf{y})$ *can be approximated using Theorem 1 as:*

$$K_{\mathrm{G}}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{\boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}_D, \mathbf{I}_D)} \left[\langle \xi_{\mathrm{G}}(\mathbf{x}), \xi_{\mathrm{G}}(\mathbf{y}) \rangle \right] \tag{26}$$

where $\xi_{\mathrm{G}}(\mathbf{x}) = \frac{1}{\sqrt{m}} \left[\sin(\boldsymbol{\omega}_1^\top \mathbf{x}), \ldots, \sin(\boldsymbol{\omega}_m^\top \mathbf{x}), \cos(\boldsymbol{\omega}^\top \mathbf{x}), \ldots, \cos(\boldsymbol{\omega}_m^\top \mathbf{x}) \right]$

Proposition 2. *The Softmax Kernel* $K_{\mathrm{SM}}(\mathbf{x}, \mathbf{y})$ *can be approximated using Theorem 1 as:*

$$K_{\mathrm{SM}}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{\boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}_D, \mathbf{I}_D)} \left[\langle \xi^{\mathrm{tg}}_{\mathrm{SM}}(\mathbf{x}), \xi^{\mathrm{tg}}_{\mathrm{SM}}(\mathbf{y}) \rangle \right] \tag{27}$$

where $\xi^{\mathrm{tg}}_{\mathrm{SM}}(\mathbf{x}) = \frac{\exp\left(\|\mathbf{x}\|^2/2\right)}{\sqrt{m}} \left[\sin(\boldsymbol{\omega}_1^\top \mathbf{x}), \ldots, \sin(\boldsymbol{\omega}_m^\top \mathbf{x}), \cos(\boldsymbol{\omega}_1^\top \mathbf{x}), \ldots, \cos(\boldsymbol{\omega}_m^\top \mathbf{x}) \right]$

We will now use the Softmax Kernel, approximated using Theorem 1, to create an efficient version of attention. Recall that attention is defined as:

$$\texttt{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\mathbf{Q}\mathbf{K}^\top\right)\mathbf{V} \tag{28}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{\ell \times D}$ encode the queries, keys, and values respectively. Note that in this assignment we focus on *unmasked*, *self*-attention and that, without loss of generality, we omit attention rescaling for clarity and ease of notation.

Eq. (28) can be also written equivalently as:

$$\texttt{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{D}^{-1}\mathbf{A}\mathbf{V} \tag{29}$$

where $\mathbf{A}$,$\mathbf{D} \in \mathbb{R}^{\ell \times \ell}$ are matrices having the following form.

$$\mathbf{A} = \exp\left(\mathbf{Q}\mathbf{K}^\top\right) \quad \mathbf{D} = \mathrm{diag}(\mathbf{A}\mathbf{1}_\ell)$$

We now use the Softmax Kernel to provide an alternative definition of self-attention, expressing the matrix $\mathbf{A} \in \mathbb{R}^{\ell \times \ell}$ through a kernel $K(\mathbf{x}, \mathbf{y})$, so that $\mathbf{A}[i, j] = K(\mathbf{Q}[i, :], \mathbf{K}[j, :])$. Using $\xi(\mathbf{x})^{\mathrm{tg}}_{\mathrm{SM}}$, we introduce a form of kernelized attention defined as follows.

$$\texttt{KernelizedAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = (\mathbf{D}')^{-1}\left(\mathbf{Q}'((\mathbf{K}')^\top \mathbf{V})\right) \in \mathbb{R}^{\ell \times d} \tag{30}$$

where $\mathbf{Q}'$, $\mathbf{K}'$, and $\mathbf{D}'$ have the following form.

$$\mathbf{Q}'[i, :] = \phi_{\mathrm{SM}}(\mathbf{Q}[i, :]^\top)^\top \tag{31}$$

$$\mathbf{K}'[i, :] = \phi_{\mathrm{SM}}(\mathbf{K}[i, :]^\top)^\top \tag{32}$$

$$\mathbf{D}' = \mathrm{diag}(\mathbf{Q}'((\mathbf{K}')^\top \mathbf{1}_\ell)) \tag{33}$$

c.1) **(2 pts)** Provide the time and space complexity of both `Attention` and `KernelizedAttention` with respect to ℓ, D, and r. Discuss why `KernelizedAttention` is advantageous compared to `Attention`.

c.2) **(1 pts)** $\xi^{\mathrm{tg}}_{\mathrm{SM}}(\mathbf{x})$ does not yield positive scores for all $\boldsymbol{\omega}$. Explain why this is problematic and provide a $\boldsymbol{\omega}$ that yields a negative score.

We now introduce a new feature map $\xi^{+}_{\mathrm{SM}}(\mathbf{x})$ that always yields positive scores.

$$\xi^{+}_{\mathrm{SM}}(\mathbf{x}) \stackrel{\mathrm{def}}{=} \frac{\exp\left(-\frac{\|\mathbf{x}^2\|}{2}\right)}{\sqrt{m}} \left[\exp(\boldsymbol{\omega}_1^\top \mathbf{x}), \ldots, \exp(\boldsymbol{\omega}_m^\top \mathbf{x}) \right] \tag{34}$$

d.1) **(3 pts)** Prove that $\xi^{+}_{\mathrm{SM}}(\mathbf{x})$ can approximate the Softmax Kernel, i.e. prove that the following identity holds.

$$K_{\mathrm{SM}}(\mathbf{x},\mathbf{y}) \stackrel{\mathrm{def}}{=} \exp\left(\mathbf{x}^{\top}\mathbf{y}\right) = \mathbb{E}_{\boldsymbol{\omega}\sim\mathcal{N}(\mathbf{0}_D,\mathbf{I}_D)}\left[\exp\left(\boldsymbol{\omega}^{\top}\mathbf{x} - \frac{||\mathbf{x}||^2}{2}\right)\exp\left(\boldsymbol{\omega}^{\top}\mathbf{y} - \frac{||\mathbf{y}||^2}{2}\right)\right] \tag{35}$$

Hint: Use the fact that:

$$(2\pi)^{-\frac{d}{2}}\int \exp\left(-\frac{||\boldsymbol{\omega}-\mathbf{v}||_2^2)}{2}\right)d\boldsymbol{\omega} = 1, \quad \forall \mathbf{v}\in\mathbb{R}^d \tag{36}$$

For each of the feature maps, we can derive a corresponding Monte Carlo estimator by taking samples from the normal distribution $\boldsymbol{\omega}\sim\mathcal{N}$. The there estimators are defined as follows.

$$\widehat{K_{\mathrm{G}}(\mathbf{x},\mathbf{y})}^{(\boldsymbol{\omega})} = \langle\xi_{\mathrm{G}}(\mathbf{x}),\xi_{\mathrm{G}}(\mathbf{y})\rangle \tag{37}$$

$$\widehat{K_{\mathrm{SM}}(\mathbf{x},\mathbf{y})}^{(\boldsymbol{\omega})}_{\mathrm{tg}} = \langle\xi^{\mathrm{tg}}_{\mathrm{SM}}(\mathbf{x}),\xi^{\mathrm{tg}}_{\mathrm{SM}}(\mathbf{y})\rangle \tag{38}$$

$$\widehat{K_{\mathrm{SM}}(\mathbf{x},\mathbf{y})}^{(\boldsymbol{\omega})}_{+} = \langle\xi^{+}_{\mathrm{SM}}(\mathbf{x}),\xi^{+}_{\mathrm{SM}}(\mathbf{y})\rangle \tag{39}$$

The goal of the next questions is to prove some properties of these estimators and discuss whether they can be effectively used to effectively compute the efficient version of attention in Eq. (30).

e.1) **(3 pts)** Prove that $\widehat{K_{\mathrm{G}}(\mathbf{x},\mathbf{y})}^{(\boldsymbol{\omega})}$ is unbiased, using the fact that:

$$\mathbb{E}\left[\cos(\boldsymbol{\omega}^{\top}(\mathbf{x}-\mathbf{y}))\right] = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2}\right)$$

e.2) **(3 pts)** Prove that:

$$\mathrm{MSE}\left(\widehat{K_{\mathrm{SM}}(\mathbf{x},\mathbf{y})}^{(\boldsymbol{\omega})}_{\mathrm{tg}}\right) = \frac{1}{2m}\exp(||\mathbf{x}+\mathbf{y}||^2)K_{\mathrm{SM}}(\mathbf{x},\mathbf{y})^{-2}(1-\exp\left(-||\mathbf{x}-\mathbf{y}||^2\right))^2 \tag{40}$$

where MSE is the mean squared error, using the fact that:

$$\mathbb{E}\left[\cos^2(\boldsymbol{\omega}^{\top}(\mathbf{x}-\mathbf{y}))\right] = \frac{1+\exp\left(-2\|\mathbf{x}-\mathbf{y}\|^2\right)}{2}$$

e.3) **(4 pts)** Prove that:

$$\mathrm{MSE}\left(\widehat{K_{\mathrm{SM}}(\mathbf{x},\mathbf{y})}^{(\boldsymbol{\omega})}_{+}\right) = \frac{1}{m}\exp(||\mathbf{x}+\mathbf{y}||^2)K_{\mathrm{SM}}(\mathbf{x},\mathbf{y})^{2}(1-\exp\left(-||\mathbf{x}+\mathbf{y}||^2\right)) \tag{41}$$

where MSE is the mean squared error, using the fact that:

$$\mathbb{E}_{\boldsymbol{\omega}\sim\mathcal{D}(\mathbf{0}_D,\mathbf{I}_D)}\left[\exp(\boldsymbol{\omega}^{\top}(\mathbf{x}+\mathbf{y}))\right] = \exp\left(\frac{||\mathbf{x}+\mathbf{y}||^2}{2}\right)$$

e.4) **(1 pts)** Solve the following limits and discuss their results.

$$\lim_{K_{SM}\to 0} \text{MSE}\left(\widehat{K_{\text{SM}}(\mathbf{x},\mathbf{y})}_{\text{tg}}^{(\boldsymbol{\omega})}\right)$$

$$\lim_{K_{SM}\to 0} \text{MSE}\left(\widehat{K_{\text{SM}}(\mathbf{x},\mathbf{y})}_{+}^{(\boldsymbol{\omega})}\right)$$

References

Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May, and Kevin Knight. Recurrent neural networks as weighted language recognizers. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2261–2271, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1205. URL https://aclanthology.org/N18-1205.

Li Du, Lucas Torroba Hennigen, Tiago Pimentel, Clara Meister, Jason Eisner, and Ryan Cotterell. A measure-theoretic characterization of tight language models, 2022. URL https://arxiv.org/abs/2212.10502.