AI Application Lecture 6
Probabilistic Language Models and Sampling

SUZUKI, Atsushi
Jing WANG

Contents

1 Introduction

1.1 Review of the Previous Lecture

In the previous lecture, we introduced the concept of a **pipeline**, which combines a neural network with other algorithms, rather than using the neural network in isolation. Specifically, we learned that in natural language processing, **tokenization** serves as the entry and exit point of the pipeline, responsible for interconverting between strings and token sequences. We also examined the formal definition and operation of Byte Pair Encoding (BPE) [2, 6], confirming its practical motivation of balancing sequence length reduction with handling unknown words.

In this lecture, based on the premise that **we have already learned how to convert inputs and outputs to token sequences**, we will rigorously formulate **how to make a neural network represent the "probability distribution of the next token" and how to generate a token sequence from it**. Since a neural network is (at a minimum) a continuous function, it is inappropriate to directly treat it as a "function that takes only discrete token sequences as its values." Therefore, we will have the **neural network represent a probabilistic language model**, and construct a **token generator** from a **pseudo-random number sequence** and the **probabilistic language model**.

1.2 Learning Outcomes for This Lecture

By the end of this lecture, students should be able to:

- Understand the rigorous framework for viewing a neural network, which is a continuous function, as a **probabilistic language model**, and obtain a probability mass function (PMF) through softmax normalization.
- Recursively construct a **token generator** from a probabilistic language model and a pseudo-random number sequence, and be able to generate token sequences by rigorously defining samplers including greedy search, beam search, temperature, top-k, top-p (nucleus [3]), and repetition penalty [4].

2 Preliminaries: Mathematical Notations

This section describes the basic mathematical notations used in this lecture.

- **Definition:**
  - $(\mathrm{LHS}) \coloneqq (\mathrm{RHS})$: Indicates that the left-hand side is defined by the right-hand side. For example, $a \coloneqq b$ indicates that a is defined as b.
- **Set:**

  - Sets are often denoted by uppercase calligraphic letters. Example: $\mathcal{A}$.
  - $x \in \mathcal{A}$: Indicates that element x belongs to set $\mathcal{A}$.
  - $\{\}$: The empty set.
  - $\{a, b, c\}$: The set consisting of elements a, b, c (roster notation).
  - $\{x \in \mathcal{A} | P(x)\}$: The set of elements in set $\mathcal{A}$ for which the proposition $P(x)$ is true (set-builder notation).
  - $|\mathcal{A}|$: The number of elements in set $\mathcal{A}$ (in this lecture, generally used only for finite sets).
  - $\mathbb{R}$: The set of all real numbers.
  - $\mathbb{R}_{>0}$: The set of all positive real numbers.
  - $\mathbb{R}_{\geq 0}$: The set of all non-negative real numbers.
  - $\mathbb{Z}$: The set of all integers.
  - $\mathbb{Z}_{>0}$: The set of all positive integers.
  - $\mathbb{Z}_{\geq 0}$: The set of all non-negative integers.
  - $[1, k]_{\mathbb{Z}}$: When k is a positive integer, $[1, k]_{\mathbb{Z}} \coloneqq \{1, 2, \ldots, k\}$, i.e., the set of integers from 1 to k. When $k = +\infty$, $[1, k]_{\mathbb{Z}} \coloneqq \mathbb{Z}_{>0}$, i.e., the set of all positive integers.

- **Function:**
  - $f : \mathcal{X} \to \mathcal{Y}$: Indicates that the function f is a map that takes an element from set $\mathcal{X}$ as input and outputs an element from set $\mathcal{Y}$.
  - $y = f(x)$: Indicates that the output of function f for the input $x \in \mathcal{X}$ is $y \in \mathcal{Y}$.

- **Vector:**
  - In this course, a vector refers to a column of numbers.
  - Vectors are denoted by bold italic lowercase letters. Example: $\boldsymbol{v}$.
  - $\boldsymbol{v} \in \mathbb{R}^n$: Indicates that the vector $\boldsymbol{v}$ is an n-dimensional real vector.
  - The i-th element of vector $\boldsymbol{v}$ is denoted by v_i.

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}. \tag{1}$$

- **Sequence:**

- Given a set $\mathcal{A}$, an $n \in \mathbb{Z}_{>0} \cup \{+\infty\}$, and a function $\boldsymbol{a} : [1, n]_{\mathbb{Z}} \to \mathcal{A}$, $\boldsymbol{a}$ is called a sequence of length n consisting of elements from set $\mathcal{A}$. When $n < +\infty$, the sequence is called a finite sequence, and when $n = \infty$, it is called an infinite sequence.
- Sequences are denoted by bold italic lowercase letters, similar to vectors. This is because a finite sequence can be seen as an extension of a real vector. In fact, a finite sequence of elements from $\mathbb{R}$ can be considered a real vector.
- For a sequence $\boldsymbol{a}$ of length n with elements from set $\mathcal{A}$, the i-th component a_i for $i \in [1, n]_{\mathbb{Z}}$ is defined as $a_i := \boldsymbol{a}(i)$.
- When $n < +\infty$, a sequence $\boldsymbol{a}$ of length n with elements from set $\mathcal{A}$ is determined by its elements $a_1, a_2, ..., a_n$, so we write $\boldsymbol{a} = (a_1, a_2, ..., a_n)$. Similarly, when $\boldsymbol{a}$ is an infinite sequence, we write $\boldsymbol{a} = (a_1, a_2, ...)$.
- The length of a sequence $\boldsymbol{a}$ is denoted by $|\boldsymbol{a}|$.

• **Matrix:**

- Matrices are denoted by bold italic uppercase letters. Example: $\boldsymbol{A}$.
- $\boldsymbol{A} \in \mathbb{R}^{m,n}$: Indicates that matrix $\boldsymbol{A}$ is an $m \times n$ real matrix.
- The element in the i-th row and j-th column of matrix $\boldsymbol{A}$ is denoted by $a_{i,j}$.

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}. \tag{2}$$

- The transpose of matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^\top$. If $\boldsymbol{A} \in \mathbb{R}^{m,n}$, then $\boldsymbol{A}^\top \in \mathbb{R}^{n,m}$, and

$$\boldsymbol{A}^\top = \begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{m,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \cdots & a_{m,n} \end{bmatrix} \tag{3}$$

- A vector is also a matrix with 1 column, and its transpose can be defined.

$$\boldsymbol{v}^\top = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \in \mathbb{R}^{1,n} \tag{4}$$

• **Tensor:**

- In this lecture, the term tensor simply refers to a multi-dimensional array. A vector can be considered a 1st-order tensor, and a matrix a 2nd-order tensor. Tensors of 3rd order or higher are denoted by underlined bold italic uppercase letters, like $\underline{\boldsymbol{A}}$.

- Students who have already learned about abstract tensors in mathematics or physics may be resistant to calling a mere multi-dimensional array a tensor. However, if we consider that the basis is always fixed to the standard basis and that we are identifying the mathematical tensor with its component representation (which becomes a multi-dimensional array), then the terminology is (for the most part) consistent.

3 Probabilistic Language Model

3.1 Motivation

A neural network f_θ typically requires its function to be at least continuous with respect to the input vector to avoid difficulties in learning. However, the desired outputs in natural language processing, whether byte sequences or token sequences, are discrete objects. A function that is both continuous and has a discrete range of possible output values must be a constant function (strictly speaking, a continuous function whose domain is a connected subset of a finite-dimensional real space and whose range is a countable subset of a finite-dimensional real space is limited to being a constant function), which is not useful for practical applications. Therefore, instead of having the neural network output the token sequence itself, the standard approach is to have the neural network output **scores for tokens (quantities analogous to unnormalized likelihoods or log-probabilities)** and interpret this as a **probabilistic language model** via the softmax function (e.g., [1]).

Before explaining probabilistic language models, let's organize the terminology and notation for tokens.

Definition 3.1 (Vocabulary and Token Sequences)**.** The set of all possible values a token can take is called the **vocabulary**, denoted by $\mathcal{V}$[a]. Hereafter, for a vocabulary of size D, we identify the vocabulary $\mathcal{V}$ with the subset of natural numbers $[1, D]_{\mathbb{Z}} = \{1, 2, \ldots, D\}$. The set of token sequences of length n can be seen as the Cartesian product of n copies of $\mathcal{V}$, written as $\mathcal{V}^n$. Note that $\mathcal{V}^0$ is the set containing only the empty token sequence $()$. Furthermore, the set of all token sequences of finite length is written as $\mathcal{V}^*$. We have $\mathcal{V}^* = \mathcal{V}^0 \cup \mathcal{V}^1 \cup \mathcal{V}^2 \cup \cdots$.

[a] In a previous lecture, the set of nodes in the neural network was also denoted by $\mathcal{V}$, but this lecture does not explicitly describe the graph structure of neural networks, so $\mathcal{V}$ should always be taken to mean the vocabulary.

Definition 3.2 (Probabilistic Language Model (Most General Form))**.** Let a finite vocabulary $\mathcal{V} = \{1, 2, \ldots, D\}$ be fixed. A **probabilistic language model** is a function $P(\cdot|\cdot)$ that, given any finite-length token sequence $\boldsymbol{t} = (t_1, \ldots, t_n) \in \mathcal{V}^n$, returns the conditional probability mass function of the next token. More formally, a two-variable function $P(\cdot|\cdot) : \mathcal{V} \times \mathcal{V}^* \to [0, 1]$ is a probabilistic language model if for any $\boldsymbol{t} \in \mathcal{V}^*$,

$$\sum_{v \in \mathcal{V}} P(v|\boldsymbol{t}) = 1 \tag{5}$$

holds.

Remark 3.1. This definition does not assume any internal representation of the model. It is a general definition that includes specific structures such as Markov properties, n-grams, and Transformers [7]. It also specifies the joint distribution of the entire sequence $P((t_1, t_2, ..., t_n)) = \prod_{i=1}^{n} P(t_i|(t_1, t_2, ..., t_{i-1}))$ (by the chain rule). While methods for constructing the joint distribution directly with a neural network are mathematically possible and could be a topic for future development, they are non-trivial to adapt for arbitrary-length outputs compared to the method of constructing conditional probability distributions, i.e., probabilistic language models.

3.2 Construction from a Neural Network

Since we want to leverage the expressive power of neural networks, we are interested in how to construct a probabilistic language model from a neural network. There are two issues to resolve.

- **Handling Variable-Length Inputs** We want to handle inputs of various lengths.
- **Handling Probability Constraints** We want to constrain the output to be non-negative and sum to 1.

3.2.1 Handling Variable-Length Inputs

Handling various input lengths is non-trivial because, for a fixed neural network, the input dimension is naively fixed, which in turn fixes the input length.

This is addressed through specific designs in the neural network architecture. To avoid losing generality by delving into specific architectural configurations, let's discuss in general terms what conditions the architecture must satisfy. There are at least two possible ways to handle variable input lengths (these are not necessarily mutually exclusive).

- **Giant Neural Network Approach** Prepare a huge neural network with a large number of parameters that accepts a very long input of length L. For shorter inputs, pad them with zeros to transform them into inputs of length L.
- **Variable-Length Input Neural Network Model Approach** Prepare an algorithm that can construct different neural networks $f_{\boldsymbol{\theta}}^{(n)}$ from the same parameters $\boldsymbol{\theta}$, depending on the input length n. These are realized through parameter sharing and recursive

structures. In this lecture, we will call a neural network $f_{\boldsymbol{\theta}}^{(\cdot)}$ with this mechanism a **variable-length input neural network**.

In reality, most applications are based on the variable-length input neural network approach. As far as the author knows, not only in natural language processing, but historically, neural networks that have shown remarkable performance in applications—such as convolutional neural networks (CNNs), recurrent neural networks (RNNs) including long-short term memory (LSTM) and gated recurrent units (GRU), and Transformers—are almost all capable of this variable neural network approach, so there are no practical issues with this method. The reason these neural networks are all variable-length is that they all have structures that can be defined recursively[1], so by simply specifying the number of repetitions n, $f_{\boldsymbol{\theta}}^{(n)}$ can be constructed. All that needs to be stored on the computer is the fixed, finite-dimensional vector $\boldsymbol{\theta}$.

Remark 3.2 (Difference between Variable-Length Input Neural Networks and an Infinite Sequence of Neural Networks)**.** A non-trivial and important point is that it is not necessary to prepare neural networks $f_{\boldsymbol{\theta}}^{(n)}$ on the computer for all $n \in \mathbb{Z}_{\geq 0}$ in advance, i.e., it is impossible to prepare an infinite sequence of neural networks on a computer due to finite memory, but this is not necessary. It is sufficient if $f_{\boldsymbol{\theta}}^{(n)}$ can be constructed only when an input is given and n is known. Therefore, it is sufficient that an algorithm exists to mechanically construct $f_{\boldsymbol{\theta}}^{(n)}$ with n as an argument.

3.2.2 Handling Probability Constraints

How can we ensure that the output is not an arbitrary vector, but a probability mass function that is non-negative and sums to 1? The method of normalization by softmax is well-known and widely used (in implementation, it may be possible to avoid the softmax calculation in some cases). Below, we show the construction of a probabilistic language model from a general parametric variable-length input function, which is not limited to neural networks, but of course, this construction can be applied to neural networks as well.

Definition 3.3 (Construction of a Probabilistic Language Model from a Variable-Length Input Function via Softmax Normalization)**.** Suppose we are given a parametric variable-length input function $f_{(\cdot)}^{(\cdot)}$ and its parameters $\boldsymbol{\theta}$, and that for any non-negative integer $n \in \mathbb{Z}_{\geq 0}$, we can always construct a function $f_{\boldsymbol{\theta}}^{(n)} : \mathcal{V}^n \to \mathbb{R}^{|\mathcal{V}|}$.

[1]On the other hand, the multi-layer perceptron, which many students learn first but is not often used in practice, does not have such a recursive structure. The fact that neural networks with recursively definable structures have been successful is a manifestation of some law of nature, and its elucidation is an interesting research topic.

Then, through normalization by the softmax function,

$$P_{f_{\boldsymbol{\theta}}^{(\cdot)}}(v \mid \boldsymbol{t}) := \frac{\exp\left(f_{\boldsymbol{\theta},v}^{(|\boldsymbol{t}|)}(\boldsymbol{t})\right)}{\sum_{u \in \mathcal{V}} \exp\left(f_{\boldsymbol{\theta},u}^{(|\boldsymbol{t}|)}(\boldsymbol{t})\right)} \quad (v \in \mathcal{V}, \boldsymbol{t} \in \mathcal{V}^*) \tag{6}$$

the defined $P_{f_{\boldsymbol{\theta}}^{(\cdot)}}(\cdot \mid \cdot)$ is a probabilistic language model.

In the above definition, the v-th component $s_{v|\boldsymbol{t}}$ of the return vector $\boldsymbol{s}_{|\boldsymbol{t}} = f_{\boldsymbol{\theta}}^{(n)}(\boldsymbol{t})$ of the function $f_{\boldsymbol{\theta}}^{(n)}$ can be regarded as the score of v given $\boldsymbol{t}$. A higher score corresponds to a larger probability mass.

In practice, a hyperparameter called temperature[2] T is often introduced to extend the construction of the probabilistic language model, allowing different probability language models to be obtained using the same variable-length neural network. Below, we describe this extended construction.

Definition 3.4 (Construction of a Probabilistic Language Model from a Variable-Length Input Function with Temperature as a Hyperparameter)**.** Suppose we are given a parametric variable-length input function $f_{(\cdot)}^{(\cdot)}$ and its parameters $\boldsymbol{\theta}$, and that for any non-negative integer $n \in \mathbb{Z}_{\geq 0}$, we can always construct a function $f_{\boldsymbol{\theta}}^{(n)} : \mathcal{V}^n \to \mathbb{R}^{|\mathcal{V}|}$.
Then, with a **temperature** $T > 0$ and normalization by the softmax function,

$$P_{f_{\boldsymbol{\theta}}^{(\cdot)},T}(v \mid \boldsymbol{t}) := \frac{\exp\left(\frac{1}{T} \cdot f_{\boldsymbol{\theta},v}^{(|\boldsymbol{t}|)}(\boldsymbol{t})\right)}{\sum_{u \in \mathcal{V}} \exp\left(\frac{1}{T} \cdot f_{\boldsymbol{\theta},u}^{(|\boldsymbol{t}|)}(\boldsymbol{t})\right)} \quad (v \in \mathcal{V}, \boldsymbol{t} \in \mathcal{V}^*) \tag{7}$$

the defined $P_{f_{\boldsymbol{\theta}}^{(\cdot)},T}(\cdot \mid \cdot)$ is a probabilistic language model.

Remark 3.3. By changing the temperature T, the spread of the probability distribution can be controlled. As $T \downarrow 0$, the probability distribution becomes sharper, and as $T \uparrow \infty$, the probability distribution approaches a uniform distribution. The name ”temperature” originates from statistical mechanics (the Boltzmann distribution, a probability distribution obtained by applying softmax to the product of the inverse of temperature and the negative energy corresponding to $f_{\boldsymbol{\theta},u}^{(|\boldsymbol{t}|)}(\boldsymbol{t})$ in the previous definition, is fundamental in statistical mechanics).

[2]Note the difference between parameters and hyperparameters in machine learning. Hyperparameters are determined independently of the training data during learning, while parameters are changed during learning according to the training data.

4 Token Generators

4.1 Two Basic Ideas: Sampling and Selecting the Maximum Probability Element

Sampling (Stochastic Generation). A probabilistic language model $P(\cdot \mid \boldsymbol{t})$ gives "the probability of each value for the next token." **Sampling** is the idea of **reproducing on a computer** the behavior of a random variable following this probability law, and generating a string by sequentially drawing tokens. The stochastic behavior provides **diversity**, which is useful in applications such as creative writing and dialogue. However, since it is difficult to obtain **true random numbers** (physical random numbers, etc.) on a general computer, a **pseudo-random number generator** is used. This generator produces a sequence with a **long period and high uniformity** using a **deterministic algorithm** and a **seed**. A typical example of a pseudo-random number generator is the **Mersenne twister**, which is widely used [5] (especially MT19937)[3].

Selecting the Maximum Probability Element (Deterministic Generation). On the other hand, for tasks where consistency is more important than diversity (such as machine translation), a natural idea is to select the **sequence with the highest probability mass** under the **joint probability** $P(t_{1:n}) = \prod_{i=1}^{n} P(t_i \mid (t_1, \ldots, t_{i-1}))$ derived from the conditional probabilities. Since exact maximization is often computationally difficult, deterministic approximate search methods such as **greedy search** and **beam search** are used. Given a probabilistic language model, these methods can essentially select a sequence deterministically.

In the following, we will look at specific algorithms for token generation based on these two ideas.

4.2 Definition: Pseudo-random Sequences and Token Generators

We want to formally define a token generator. Since we may want the token generator to have stochastic behavior, we start with the definition of pseudo-random numbers.

Definition 4.1 (Pseudo-random sequence)**.** Given a fixed initial seed $s \in \mathbb{Z}$, a generator PRNG that deterministically returns an infinite sequence $\boldsymbol{u}(s) = (u_1, u_2, \ldots)$ is called a pseudo-random number generator. Each u_i is a real number in $[0, 1)$ (including finite-precision floating-point numbers).

Now, we define a token generator in a way that can depend on a pseudo-random number generator.

[3]Note that while the Mersenne twister is excellent for most purposes, it is not suitable for cryptographic applications.

Definition 4.2 (Token Generator (General Form))**.** Let a probabilistic language model P and a PRNG be fixed. A **token generator** is a deterministic function

$$\mathsf{Gen}_{(P,\,\mathsf{PRNG})} : \mathcal{V}^* \times \mathbb{Z} \to \mathcal{V}^* \tag{8}$$

that, for any input sequence $\boldsymbol{t}_{\mathrm{in}} \in \mathcal{V}^*$ and seed s, returns an output sequence $\boldsymbol{t}_{\mathrm{out}} \in \mathcal{V}^*$. Internally, it sequentially consumes $\boldsymbol{u}(s)$ while appending tokens according to an explicit rule based on P, and stops when a **termination condition** is met.

Remark 4.1. If the initial seed s is fixed, the output is **completely deterministic**. The stochastic behavior is solely handled by the PRNG, and the generation rule itself is mathematically deterministic.

4.3 Sampling

Definition 4.3 (Sampling using a Probabilistic Language Model)**.** Given a vocabulary $\mathcal{V} = \{1, \ldots, D\}$, an input sequence $\boldsymbol{t}^{(0)} = \boldsymbol{t}_{\mathrm{in}}$, and a termination predicate Stop $: \mathcal{V}^* \to \{0, 1\}$ (e.g., occurrence of an EOS token or reaching a maximum length). For $i = 0, 1, 2, \ldots$:

- If Stop$(\boldsymbol{t}^{(i)}) = 1$, stop and return $\boldsymbol{t}_{\mathrm{out}} = \boldsymbol{t}^{(i)}$.
- Calculate the cumulative distribution $F^{(i)}(v) \coloneqq \sum_{u \le v} P\left(u | \boldsymbol{t}^{(i)}\right)$ for all $v \in \mathcal{V}$.
- $t_{i+1} \coloneqq \min\{\, v \in \mathcal{V} \mid F^{(i)}(v) > u_{i+1} \,\}$.
- Repeat with $\boldsymbol{t}^{(i+1)} \coloneqq (\boldsymbol{t}^{(i)},\, t_{i+1})$.

The function defined by this procedure is denoted as $\mathsf{NaiveSample}_{(P,\,\mathsf{PRNG},\,\mathsf{Stop})}$.

Proposition 4.1 (Consistency Check)**.** $\mathsf{NaiveSample}_{(P,\,\mathsf{PRNG},\,\mathsf{Stop})}$ satisfies the definition of a token generator.

Proof. The rule used in each iteration is a deterministic map uniquely determined by $\boldsymbol{t}^{(i)}$ and u_{i+1}, and it terminates in a finite time due to Stop (guaranteed if a maximum length is set). Thus, it conforms to the definition. □

Example 4.1 (Two Steps of Naive Sampling)**.** Let $\mathcal{V} = \{1, 2, 3\}$, input be the empty sequence $()$, and termination be at length 2. The probability distributions are given as follows:

$$\begin{aligned}
P(\cdot \mid ()) &= (0.50,\ 0.30,\ 0.20), \\
P(\cdot \mid (1)) &= (0.30,\ 0.40,\ 0.30), \\
P(\cdot \mid (2)) &= (0.10,\ 0.20,\ 0.70), \\
P(\cdot \mid (3)) &= (0.60,\ 0.10,\ 0.30).
\end{aligned}$$

Assume $u_1 = 0.68,\ u_2 = 0.72$ are obtained from the seed.

- Step 1: $\boldsymbol{t}^{(0)} = ()$. The cumulative distribution based on $P(\cdot|())$ is $(0.50, 0.80, 1.00)$. Since $u_1 = 0.68$ falls in the interval $(0.50, 0.80]$, $t_1 = 2$. We get $\boldsymbol{t}^{(1)} = (2)$.
- Step 2: $\boldsymbol{t}^{(1)} = (2)$. The cumulative distribution based on $P(\cdot|(2))$ is $(0.10, 0.30, 1.00)$. Since $u_2 = 0.72$ falls in $(0.30, 1.00]$, $t_2 = 3$. We get $\boldsymbol{t}^{(2)} = (2, 3)$.

The length is now 2, so we stop. The output sequence is $(2, 3)$.

Exercise 4.1 (Manual Calculation Exercise (Naive Sampling))**.** Let $\mathcal{V} = \{1, 2, 3\}$, input be the empty sequence $()$, and stop at length 2. The probability distributions are given as follows:

$$\begin{aligned} P(\cdot \mid ()) &= (0.40,\ 0.40,\ 0.20), \\ P(\cdot \mid (1)) &= (0.25,\ 0.25,\ 0.50), \\ P(\cdot \mid (2)) &= (0.10,\ 0.70,\ 0.20), \\ P(\cdot \mid (3)) &= (0.50,\ 0.30,\ 0.20). \end{aligned}$$

Given $u_1 = 0.41,\ u_2 = 0.20$ from the seed, find the output sequence (explicitly show the cumulative distribution and interval for each step).

Solution. Step 1: $\boldsymbol{t}^{(0)} = ()$. The cumulative distribution based on $P(\cdot|())$ is $(0.40, 0.80, 1.00)$. Since $u_1 = 0.41 \in (0.40, 0.80]$, $t_1 = 2$. We get $\boldsymbol{t}^{(1)} = (2)$. Step 2: $\boldsymbol{t}^{(1)} = (2)$. The cumulative distribution based on $P(\cdot|(2)) = (0.1, 0.7, 0.2)$ is $(0.10, 0.80, 1.00)$. Since $u_2 = 0.20 \in (0.10, 0.80]$, $t_2 = 2$. We get $\boldsymbol{t}^{(2)} = (2, 2)$. The length is now 2, so we stop. The output sequence is $(2, 2)$.

4.4 Modifying Probabilistic Language Models and Sampling Based on Them

In actual applications, the probabilistic language model naturally obtained from a neural network is often modified rather than used as is. The motivations are as follows:

- **Preventing Repetition** We want to avoid monotonous strings caused by repetition.
- **Excluding Low-Score Tokens** We want to exclude tokens with excessively low scores, even if they have a slight probability.

- To **prevent repetition**, the probabilistic language model is modified with the following strategy:
  - Divide the scores of previously seen tokens by a penalty value called **repetition penalty** to relatively disadvantage them and suppress repetition [4].
- To **exclude low-score tokens**, the probabilistic language model is modified with the following strategies:

- Lower the **temperature** T below 1 to reduce the diversity of the probability distribution (although, theoretically, a strategy to increase diversity by setting T above 1 is possible).
- Use **top-k**, which excludes all but the top k candidate tokens with the highest probability mass (equivalent to the highest scores) by setting their probability mass to 0.
- Use **top-p** to exclude all but the top candidate tokens in the **nucleus**, which are the tokens whose cumulative probability sum, in descending order, reaches p, by setting their probability mass to 0 [3].

These are commonly used techniques.

Let's introduce an algorithm for constructing a conditional probability mass function that applies these in the **common Hugging Face order** (repetition → temperature → top-k → top-p). Strictly speaking:

Definition 4.4 (Construction of Conditional Probability Mass Function Used in Hugging Face Sampling)**.** Let the unnormalized scores for a history $\boldsymbol{t}$ be $\boldsymbol{s}_{|\boldsymbol{t}} \in \mathbb{R}^D$. Fix the set of previously seen tokens $H(\boldsymbol{t}) \subseteq \mathcal{V}$, temperature $T > 0$, repetition penalty factor $\lambda > 0$, $k \in \mathbb{Z}_{>0} \cup \{\infty\}$, and $p \in (0, 1]$. $P(\cdot|\boldsymbol{t})$ is constructed as follows:

1. (**repetition**[a]) $s^{\mathrm{rep}}_{v|\boldsymbol{t}} \leftarrow \begin{cases} s_{v|\boldsymbol{t}} & \text{if } v \notin H(\boldsymbol{t}), \\ \frac{1}{\lambda} s_{v|\boldsymbol{t}} & \text{if } v \in H(\boldsymbol{t}) \text{ and } s_{v|\boldsymbol{t}} \geq 0, \\ \lambda s_{v|\boldsymbol{t}} & \text{if } v \in H(\boldsymbol{t}) \text{ and } s_{v|\boldsymbol{t}} < 0. \end{cases}$
2. (**temperature**) $s^{\mathrm{rep},T}_{v|\boldsymbol{t}} \leftarrow \dfrac{s^{\mathrm{rep}}_{v|\boldsymbol{t}}}{T}$.
3. (**top-k**) Consider scores below the top k as $-\infty$, making their probability 0 (if $k = \infty$, this is inactive).
4. (**softmax**) $\pi(v) := \dfrac{\exp\left(s^{\mathrm{rep},T}_{v|\boldsymbol{t}}\right)}{\sum_u \exp\left(s^{\mathrm{rep},T}_{u|\boldsymbol{t}}\right)}$.
5. (**top-p (nucleus)**) Sort π in descending order and take the cumulative sum. The set of elements up to and including the one that **first exceeds** the threshold p forms the nucleus $\mathcal{N}$. Set the probabilities outside $\mathcal{N}$ to 0 and renormalize within $\mathcal{N}$ to obtain $P(\cdot|\boldsymbol{t})$. That is, $P(v|\boldsymbol{t}) = \dfrac{\pi(v)}{\sum_{u \in \mathcal{N}} \pi(u)}$.

[a]This operation is mathematically unnaturally complex. Simply subtracting a constant from the scores would be simpler and would provide the same operation for the same probability mass function, regardless of any constant bias in the score function. However, in practice, the operation defined here is used and is said to be useful. The definition of this operation (multiplicative rather than additive modification) seems to originate from [4].

Definition 4.5 (Hugging Face Sampling Algorithm)**.** The token generator constructed by Definition 4.3, using the conditional probability mass function $P(\cdot|\boldsymbol{t})$ from Definition 4.4 at each time step, is denoted as $\text{Sample}^{\text{HF}}_{(P,\,\text{PRNG},T,\lambda,k,p,\text{Stop})}$.

Example 4.2 (Hugging Face Style Sampling (Two Steps, Numerical Calculation))**.** Let $\mathcal{V} = \{1, 2, 3, 4\}$, input be the empty sequence $()$, and stop at length 2. The initial and conditional scores are given as follows:

$$\begin{aligned}
\boldsymbol{s}_{|()} &= (2.0,\ 1.0,\ 0.0,\ -1.0),\\
\boldsymbol{s}_{|(1)} &= (1.5,\ 0.0,\ 0.5,\ -0.2),\\
\boldsymbol{s}_{|(2)} &= (-0.1,\ 2.2,\ 0.8,\ 0.1),\\
\boldsymbol{s}_{|(3)} &= (0.9,\ 0.8,\ 1.1,\ 1.3),\\
\boldsymbol{s}_{|(4)} &= (-1.0,\ -0.5,\ 0.5,\ 1.5).
\end{aligned}$$

Hyperparameters: $T = 0.5,\ \lambda = 1.2,\ k = 3,\ p = 0.9.$

1. **Step 1** ($\boldsymbol{t} = ()$, set of seen tokens $H(()) = \{\}$)
   (a) (rep) None $\Rightarrow (2.0, 1.0, 0.0, -1.0)$.
   (b) (temp) Multiply by $1/T = 2 \Rightarrow (4.0, 2.0, 0.0, -2.0)$.
   (c) (top-k) Keep top 3 $\Rightarrow (4.0, 2.0, 0.0, -\infty)$.
   (d) (softmax) $\pi \approx (0.8668,\ 0.1173,\ 0.0159,\ 0)$.
   (e) (top-p) For $p = 0.9$, descending cumulative sum is $(0.8668, 0.9841, \dots)$. The nucleus is $\{1, 2\}$. Renormalize to get $\tilde{\pi} \approx (0.8808,\ 0.1192,\ 0,\ 0)$.
   (f) (inverse transform) If $u_1 = 0.50$ from the seed, then $t_1 = 1$.

2. **Step 2** ($\boldsymbol{t} = (1)$, set of seen tokens $H((1)) = \{1\}$)
   (a) (rep) Multiply the 1st component of score $\boldsymbol{s}_{|(1)}$ by $1/\lambda$: $(1.5/1.2,\ 0.0,\ 0.5,\ -0.2) = (1.25,\ 0.0,\ 0.5,\ -0.2)$.
   (b) (temp) Multiply by $1/T = 2$: $(2.5,\ 0.0,\ 1.0,\ -0.4)$.
   (c) (top-k) Keep top 3: $(2.5,\ 0.0,\ 1.0,\ -\infty)$.
   (d) (softmax) $\pi \approx (0.7662,\ 0.0629,\ 0.1710,\ 0)$.
   (e) (top-p) For $p = 0.9$, descending cumulative sum is $(0.7662, 0.9372, \dots)$, so the nucleus is $\{1, 3\}$. Renormalize to get $\tilde{\pi} \approx (0.8176,\ 0,\ 0.1824,\ 0)$.
   (f) (inverse transform) If $u_2 = 0.90$, then $t_2 = 3$ (since $0.8176 < 0.90 \le 1$).

The output sequence is $(1, 3)$. Numbers are rounded to about 4 decimal places.

Exercise 4.2 (Manual Calculation of Hugging Face Style Sampling)**.** Let $\mathcal{V} = \{1, 2, 3\}$, stop at length 2. The score vectors are given as follows:

$$\begin{aligned} s_{|()} &= (1.2,\ 0.7,\ -0.1), \\ s_{|(1)} &= (0.6,\ 0.4,\ 0.0), \\ s_{|(2)} &= (0.9,\ 1.1,\ 0.2), \\ s_{|(3)} &= (0.1,\ 0.2,\ 1.5). \end{aligned}$$

Hyperparameters: $T = 0.8,\ \lambda = 1.1,\ k = 2,\ p = 0.85$. Vocabulary order is $1 < 2 < 3$. Given $u_1 = 0.60,\ u_2 = 0.30$ from the seed, find (t_1, t_2) (explicitly show processor/warper/softmax/nucleus/inverse transform for each step).

Solution. Step 1 ($\boldsymbol{t} = (),\ H(()) = \{\}$)

1. (rep): No change. $s_{|()} = (1.2, 0.7, -0.1)$.
2. (temp): Multiply by $1/T = 1.25 \Rightarrow (1.5, 0.875, -0.125)$.
3. (top-k): Keep top 2 $(1.5, 0.875)$, the 3rd becomes $-\infty$.
4. (softmax): $\pi \propto (e^{1.5}, e^{0.875}, 0) \approx (4.4817, 2.3989, 0)$. Normalize to get $\pi \approx (0.6518, 0.3482, 0)$.
5. (top-p): $p = 0.85$. Descending cumulative sum is $(0.6518, 1.0)$. The nucleus is $\{1, 2\}$. No change after renormalization.
6. (inverse transform): Cumulative distribution is $(0.6518, 1.0)$. Since $u_1 = 0.60 \le 0.6518$, $t_1 = 1$.

Step 2 ($\boldsymbol{t} = (1),\ H((1)) = \{1\}$)

1. (rep): $s_{|(1)} = (0.6, 0.4, 0.0)$. Multiply 1st component by $1/\lambda = 1/1.1 \Rightarrow (0.5455, 0.4, 0.0)$.
2. (temp): Multiply by $1/T = 1.25 \Rightarrow (0.6818, 0.5, 0.0)$.
3. (top-k): Keep top 2 $(0.6818, 0.5)$, the 3rd becomes $-\infty$.
4. (softmax): $\pi \propto (e^{0.6818}, e^{0.5}, 0) \approx (1.9774, 1.6487, 0)$. Normalize to get $\pi \approx (0.545, 0.455, 0)$.
5. (top-p): $p = 0.85$. Descending cumulative sum is $(0.545, 1.0)$. The nucleus is $\{1, 2\}$. No change after renormalization.
6. (inverse transform): Cumulative distribution is $(0.545, 1.0)$. Since $u_2 = 0.30 \le 0.545$, $t_2 = 1$.

The final output sequence is $(1, 1)$.

4.5 Selecting the Maximum Probability Element: Greedy Search and Its Rigorous Definition

Definition 4.6 (Greedy decoding)**.** Given a vocabulary $\mathcal{V}$, an input sequence $\boldsymbol{t}^{(0)} = \boldsymbol{t}_{\text{in}}$, and a termination predicate Stop. At each step,

$$t_{i+1} := \arg\max_{v\in\mathcal{V}} P\big(v \mid \boldsymbol{t}^{(i)}\big), \qquad \boldsymbol{t}^{(i+1)} := (\boldsymbol{t}^{(i)}, t_{i+1}), \tag{9}$$

Ties are broken by a predetermined rule (e.g., vocabulary order). Stop when Stop is satisfied.

Proposition 4.2 (Consistency as a Generator)**.** Greedy search satisfies the definition of a token generator (a deterministic rule that does not consume random numbers).

Proof. The selection at each step is deterministically calculated from $\boldsymbol{t}^{(i)}$. Termination follows Stop. □

Example 4.3 (Greedy Search (Two Steps, Numerical))**.** Let $\mathcal{V} = \{1, 2, 3\}$, stop at length 2. The probability distributions are given as follows:

$$\begin{aligned} P(\cdot \mid ()) &= (0.55,\ 0.40,\ 0.05),\\ P(\cdot \mid (1)) &= (0.20,\ 0.70,\ 0.10),\\ P(\cdot \mid (2)) &= (0.30,\ 0.30,\ 0.40),\\ P(\cdot \mid (3)) &= (0.80,\ 0.10,\ 0.10). \end{aligned}$$

In the first step, select 1, the component with the highest probability in $P(\cdot|())$, so $t_1 = 1$. In the second step, select 2, the component with the highest probability in $P(\cdot|(1))$, so $t_2 = 2$. Thus, the output is $(1, 2)$.

Exercise 4.3 (Manual Calculation of Greedy Search)**.** Let $\mathcal{V} = \{1, 2\}$, stop at length 2. The probability distributions are given as follows:

$$\begin{aligned} P(\cdot \mid ()) &= (0.51,\ 0.49),\\ P(\cdot \mid (1)) &= (0.40,\ 0.60),\\ P(\cdot \mid (2)) &= (0.70,\ 0.30). \end{aligned}$$

Find (t_1, t_2).

Solution. In the first step, select 1, the component with the highest probability in $P(\cdot|())$, so $t_1 = 1$. In the second step, select 2, the component with the highest probability in $P(\cdot|(1))$, so $t_2 = 2$. Thus, the output is $(1, 2)$.

4.6 Limitations of Greedy Search and Beam Search (from the Perspective of Joint Probability)

Greedy search does not guarantee maximization of the joint probability. The joint probability of a two-step sequence is $P(t_1, t_2) = P(t_1 \mid ()) \cdot P(t_2 \mid (t_1))$. In the following example, greedy search fails to select the best sequence.

Example 4.4 (Greedy Fails, Beam Search Succeeds)**.** Let $\mathcal{V} = \{A, B\}$, stop at length 2. The probability distributions are given as follows:

$$\begin{aligned} P(\cdot \mid ()) &= (0.60,\ 0.40), \\ P(\cdot \mid (A)) &= (0.50,\ 0.50), \\ P(\cdot \mid (B)) &= (0.95,\ 0.05). \end{aligned}$$

- **Greedy Search**: In the first step, select A with the highest probability ($P(A|()) = 0.60$). In the second step, under $P(\cdot|(A))$, the probabilities for A and B are tied (0.50). Assuming we choose A, the joint probability of the output (A, A) is $P(A, A) = 0.60 \times 0.50 = 0.30$.
- **True Best Sequence**: Calculating all possibilities,
  - $P(A, A) = 0.60 \times 0.50 = 0.30$
  - $P(A, B) = 0.60 \times 0.50 = 0.30$
  - $P(B, A) = 0.40 \times 0.95 = 0.38$
  - $P(B, B) = 0.40 \times 0.05 = 0.02$

  The joint probability of (B, A), 0.38, is the maximum.

Thus, greedy search fails to find the optimal solution.

Definition 4.7 (Beam Search (Beam width B, max length L))**.** Initialize beam $\mathcal{B}_0 = \{((), 0)\}$ (sequence and log-likelihood). For $i = 1, \ldots, L$:

$$\begin{aligned} C_i := \Big\{ (\boldsymbol{t}',\ \ell') \ \Big|\ (\boldsymbol{t}, \ell) \in \mathcal{B}_{i-1},\ v \in \mathcal{V},\ \boldsymbol{t}' = (\boldsymbol{t}, v), \qquad (10) \\ \ell' = \ell + \log P(v \mid \boldsymbol{t}) \Big\}. \qquad (11) \end{aligned}$$

Select the top B candidates from C_i based on ℓ' to form $\mathcal{B}_i$ (can be pruned by EOS). After finishing, return the sequence with the highest ℓ.

Proposition 4.3 (Consistency as a Generator)**.** Beam search is a deterministic algorithm that satisfies the definition of a token generator (if a rule for tie-breaking is fixed).

Proof. The expansion and selection at each iteration are deterministically calculated from $\boldsymbol{t}$ and P. Termination is given by the maximum length or EOS. □

Example 4.5 (Beam search with width $B = 2$ returns the optimal sequence in the previous example)**.** Let $L = 2$.

- $i = 1$: Candidates are $\{(A, \log 0.60), (B, \log 0.40)\}$. With width 2, both are kept. $\mathcal{B}_1 = \{(A, \log 0.60), (B, \log 0.40)\}$.
- $i = 2$:
  - Expand $(A, \log 0.60)$: The log-likelihood of (A, A) is $\log 0.60 + \log 0.50 = \log 0.30$. Similarly for (A, B), it is $\log 0.30$.
  - Expand $(B, \log 0.40)$: The log-likelihood of (B, A) is $\log 0.40 + \log 0.95 = \log 0.38$. For (B, B), it is $\log 0.40 + \log 0.05 = \log 0.02$.
- The log-likelihoods of the elements in the candidate set C_2 are $\{\log 0.30, \log 0.30, \log 0.38, \log 0.02\}$. We keep the top 2 for $\mathcal{B}_2$. The maximum is (B, A) with $\log 0.38$.

Thus, beam search returns the sequence with the highest joint probability, (B, A).

Exercise 4.4 (Comparison Exercise for Beam Search)**.** Let $\mathcal{V} = \{X, Y\}$, $L = 2$. The probability distributions are given as follows:

$$\begin{aligned} P(\cdot \mid ()) &= (0.55,\ 0.45), \\ P(\cdot \mid (X)) &= (0.50,\ 0.50), \\ P(\cdot \mid (Y)) &= (0.80,\ 0.20). \end{aligned}$$

(1) Find the output sequence and joint probability for greedy search. (2) Find the output sequence and joint probability for beam search with width $B = 2$, and compare.

Solution. (1) **Greedy Search**: In the first step, select X with the highest probability ($P(X|()) = 0.55$). In the second step, under $P(\cdot|(X))$, the probabilities for X and Y are tied (0.50). Assuming we choose X, the joint probability of the output (X, X) is $0.55 \times 0.50 = 0.275$.
(2) **Beam Search (**$B = 2$**)**:

- $i = 1$: Candidates are $\{(X, \log 0.55), (Y, \log 0.45)\}$. Both are kept.
- $i = 2$:
  - Expand from X: $P(X, X) = 0.55 \times 0.50 = 0.275$, $P(X, Y) = 0.55 \times 0.50 = 0.275$.
  - Expand from Y: $P(Y, X) = 0.45 \times 0.80 = 0.36$, $P(Y, Y) = 0.45 \times 0.20 = 0.09$.
- The maximum among the four candidate joint probabilities $\{0.275, 0.275, 0.36, 0.09\}$ is 0.36.

The output of beam search is (Y, X) with a joint probability of 0.36. It found a sequence with a higher probability than the greedy search result (0.275).

4.7 Positioning as Token Generators

Proposition 4.4 (Positioning)**.** Greedy search, the sampling methods from the previous section (naive and Hugging Face style), and beam search are all special cases of **token generators** implemented within the framework of $\mathrm{Gen}_{(P,\,\mathrm{PRNG})}$ (consuming PRNG if random numbers are used, otherwise not).

Proof. Each algorithm maps to the next token via a deterministic rule (referencing $\boldsymbol{u}(s)$ if necessary) and terminates according to a given Stop. This follows the definition. □

5 Summary

- Since neural networks return continuous outputs, it is natural to use them as **probabilistic language models** rather than having them directly return discrete tokens. By obtaining a PMF via softmax, the joint distribution of a sequence is determined by the chain rule.
- Within the framework of a **sequential token generator** with an explicit pseudo-random number sequence and termination predicate, we can rigorously and uniformly define greedy search, beam search, and probabilistic sampling (including temperature, top-k, top-p, and repetition penalty).

6 Next Time

Next time, we will discuss the **embedding layer**, which is used at the initial stage of many probabilistic language models, focusing on the properties of the mapping between the vocabulary and a continuous vector space, the learning rules, and its statistical interpretation.

References

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. Journal of Machine Learning Research, 3:1137–1155, 2003.

[2] Philip Gage. A new algorithm for data compression. C Users Journal, 12(2):23–38, 1994.

[3] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In Proceedings of ICLR 2020, 2020.

[4] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. arXiv preprint arXiv:1909.05858, 2019.

[5] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation (TOMACS), 8(1):3–30, 1998.

[6] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Proceedings of ACL 2016, 2016.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Proceedings of NeurIPS 2017, 2017.