AI Application Lecture 6

Probabilistic Language Models and Sampling

SUZUKI, Atsushi
Jing WANG

Outline

Introduction

1.1 Review of the Previous Lecture

In the previous lecture, we discussed the **pipeline** concept, where a neural network is combined with other algorithms.

1.1 Review of the Previous Lecture

In the previous lecture, we discussed the **pipeline** concept, where a neural network is combined with other algorithms.

- We learned that **tokenization** is the crucial entry and exit point for natural language processing pipelines, converting between raw text strings and sequences of tokens.

1.1 Review of the Previous Lecture

In the previous lecture, we discussed the **pipeline** concept, where a neural network is combined with other algorithms.

- We learned that **tokenization** is the crucial entry and exit point for natural language processing pipelines, converting between raw text strings and sequences of tokens.
- We examined Byte Pair Encoding (BPE), understanding its motivation: balancing sequence length reduction against the need to handle unknown words.

1.1 From Tokens to Probabilities

Building on what we've learned, today we address the next logical step.

1.1 From Tokens to Probabilities

Building on what we've learned, today we address the next logical step.

- **Premise:** We now know how to convert text to and from token sequences.

1.1 From Tokens to Probabilities

Building on what we've learned, today we address the next logical step.

- **Premise:** We now know how to convert text to and from token sequences.
- **Question:** How do we make a neural network generate a probable sequence of tokens?

1.1 From Tokens to Probabilities

Building on what we've learned, today we address the next logical step.

- **Premise:** We now know how to convert text to and from token sequences.
- **Question:** How do we make a neural network generate a probable sequence of tokens?

A neural network is a continuous function, but a token sequence is a discrete object.

1.1 From Tokens to Probabilities

Building on what we've learned, today we address the next logical step.

- **Premise:** We now know how to convert text to and from token sequences.
- **Question:** How do we make a neural network generate a probable sequence of tokens?

A neural network is a continuous function, but a token sequence is a discrete object.

We will have the neural network represent a **probabilistic language model** and use that model along with a pseudo-random number generator to build a **token generator**.

1.2 Learning Outcomes for This Lecture

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

- Understand the framework of viewing a continuous neural network as a **probabilistic language model** by using softmax normalization to obtain a probability mass function.

1.2 Learning Outcomes for This Lecture

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

- Understand the framework of viewing a continuous neural network as a **probabilistic language model** by using softmax normalization to obtain a probability mass function.
- Recursively construct a **token generator** from a probabilistic language model and a pseudo-random number sequence.

1.2 Learning Outcomes for This Lecture

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

- Understand the framework of viewing a continuous neural network as a **probabilistic language model** by using softmax normalization to obtain a probability mass function.
- Recursively construct a **token generator** from a probabilistic language model and a pseudo-random number sequence.
- Rigorously define and generate token sequences using various samplers, including greedy search, beam search, and methods involving temperature, top-k, top-p, and repetition penalty.

Preliminaries: Mathematical Notations

2. Preliminaries: Mathematical Notations

Set:

- Sets: $\mathcal{A}$
- Membership: $x \in \mathcal{A}$
- Empty set: $\{\}$
- Roster notation: $\{a, b, c\}$
- Set-builder: $\{x \in \mathcal{A} | P(x)\}$
- Cardinality: $|\mathcal{A}|$
- Real numbers: $\mathbb{R}, \mathbb{R}_{>0}, \mathbb{R}_{\geq 0}$
- Integers: $\mathbb{Z}, \mathbb{Z}_{>0}, \mathbb{Z}_{\geq 0}$
- Integer range: $[1, k]_{\mathbb{Z}} := \{1, \ldots, k\}$

Function:

- $f : \mathcal{X} \to \mathcal{Y}$: f maps from $\mathcal{X}$ to $\mathcal{Y}$.
- $y = f(x)$: The output of f for input x.

Definition:

- $(\mathrm{LHS}) := (\mathrm{RHS})$: Left side is defined by the right side.

Sequence: Denoted by $\boldsymbol{a} = (a_1, a_2, \dots)$.

- A function $\boldsymbol{a} : [1, n]_{\mathbb{Z}} \to \mathcal{A}$.
- Length is denoted by $|\boldsymbol{a}|$.

Vector: Denoted by $\boldsymbol{v}$.

- A column of numbers, $\boldsymbol{v} \in \mathbb{R}^n$.
- i-th element is v_i.

Matrix: Denoted by $\boldsymbol{A}$.

- $m \times n$ matrix: $\boldsymbol{A} \in \mathbb{R}^{m,n}$.
- (i, j)-th element is $a_{i,j}$.
- Transpose: $\boldsymbol{A}^\top$.

Tensor: Denoted by $\underline{\boldsymbol{A}}$.

- Simply a multi-dimensional array.
- Vector $\to$ 1st-order, Matrix $\to$ 2nd-order.

Probabilistic Language Model

3.1 Motivation

A neural network f_θ is typically a continuous function.

3.1 Motivation

A neural network f_θ is typically a continuous function.

However, the desired output in natural language processing (a token sequence) is a discrete object.

3.1 Motivation

A neural network f_θ is typically a continuous function.

However, the desired output in natural language processing (a token sequence) is a discrete object.

A function that is both continuous and has a discrete range must be a constant function, which is not useful.

3.1 Motivation

A neural network f_θ is typically a continuous function.

However, the desired output in natural language processing (a token sequence) is a discrete object.

A function that is both continuous and has a discrete range must be a constant function, which is not useful.

Solution: Instead of outputting the token sequence itself, the neural network outputs **scores for tokens**. These scores are then converted into probabilities via the softmax function, defining a **probabilistic language model**.

Let's formalize our terms.

Definition (Vocabulary and Token Sequences)

The set of all possible values a token can take is called the **vocabulary**, denoted by $\mathcal{V}$.

Let's formalize our terms.

Definition (Vocabulary and Token Sequences)

The set of all possible values a token can take is called the **vocabulary**, denoted by $\mathcal{V}$.

- For a vocabulary of size D, we identify $\mathcal{V}$ with the set of integers $\{1, 2, \ldots, D\}$.

3.1 Motivation

Let's formalize our terms.

Definition (Vocabulary and Token Sequences)

The set of all possible values a token can take is called the **vocabulary**, denoted by $\mathcal{V}$.

- For a vocabulary of size D, we identify $\mathcal{V}$ with the set of integers $\{1, 2, \ldots, D\}$.
- The set of token sequences of length n is $\mathcal{V}^n$.

Let's formalize our terms.

Definition (Vocabulary and Token Sequences)

The set of all possible values a token can take is called the **vocabulary**, denoted by $\mathcal{V}$.

- For a vocabulary of size D, we identify $\mathcal{V}$ with the set of integers $\{1, 2, \ldots, D\}$.
- The set of token sequences of length n is $\mathcal{V}^n$.
- The set of all token sequences of any finite length is $\mathcal{V}^* = \mathcal{V}^0 \cup \mathcal{V}^1 \cup \mathcal{V}^2 \cup \cdots$.

Now we can define the central concept.

Definition (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.

Now we can define the central concept.

Definition (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, for any $\boldsymbol{t} \in \mathcal{V}^*$, the following must hold:

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

Remark

This definition is very general. It does not assume any specific model structure like Markov properties, n-grams, or Transformers [4].

Remark

This definition is very general. It does not assume any specific model structure like Markov properties, n-grams, or Transformers [4].

It also specifies the joint distribution of an entire sequence via the chain rule of probability:

$$P((t_1, \ldots, t_n)) = \prod_{i=1}^{n} P(t_i | (t_1, \ldots, t_{i-1}))$$

3.1 Motivation

Remark

This definition is very general. It does not assume any specific model structure like Markov properties, n-grams, or Transformers [4].

It also specifies the joint distribution of an entire sequence via the chain rule of probability:

$$P((t_1, \ldots, t_n)) = \prod_{i=1}^{n} P(t_i | (t_1, \ldots, t_{i-1}))$$

Constructing the joint distribution directly is possible but harder to adapt for arbitrary-length outputs compared to modeling conditional probabilities.

3.2 Construction from a Neural Network

To construct a probabilistic language model from a neural network, we must solve two issues:

- **Handling Variable-Length Inputs:** How can a fixed neural network architecture process token sequences of different lengths?

3.2 Construction from a Neural Network

To construct a probabilistic language model from a neural network, we must solve two issues:

- **Handling Variable-Length Inputs:** How can a fixed neural network architecture process token sequences of different lengths?
- **Handling Probability Constraints:** How do we ensure the network's output is a valid probability distribution (non-negative and sums to 1)?

3.2.1 Handling Variable-Length Inputs

This is addressed through special neural network architectures. Two general approaches exist:

- **Giant Neural Network Approach:** Use a huge network that accepts a very long input of fixed length L. Pad shorter inputs to match this length.

[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.

3.2.1 Handling Variable-Length Inputs

This is addressed through special neural network architectures. Two general approaches exist:

- **Giant Neural Network Approach:** Use a huge network that accepts a very long input of fixed length L. Pad shorter inputs to match this length.
- **Variable-Length Input Neural Network Model Approach:** Use an architecture with parameter sharing and recursive structures (like RNNs or Transformers). This allows[1] constructing a network $f_{\boldsymbol{\theta}}^{(n)}$ tailored to the input length n from a single set of parameters $\boldsymbol{\theta}$.

[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.

3.2.1 Handling Variable-Length Inputs

This is addressed through special neural network architectures. Two general approaches exist:

- **Giant Neural Network Approach:** Use a huge network that accepts a very long input of fixed length L. Pad shorter inputs to match this length.
- **Variable-Length Input Neural Network Model Approach:** Use an architecture with parameter sharing and recursive structures (like RNNs or Transformers). This allows[1] constructing a network $f_{\boldsymbol{\theta}}^{(n)}$ tailored to the input length n from a single set of parameters $\boldsymbol{\theta}$.

In practice, most successful models (CNNs, RNNs, Transformers) use the second approach.

[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.

Remark (Difference between Variable-Length Input NN and an Infinite Sequence of NNs)

A crucial point is that we don't need to store an infinite number of networks $f_{\boldsymbol{\theta}}^{(n)}$ for all possible lengths n.

Remark (Difference between Variable-Length Input NN and an Infinite Sequence of NNs)

A crucial point is that we don't need to store an infinite number of networks $f_{\boldsymbol{\theta}}^{(n)}$ for all possible lengths n.

Instead, we only need an **algorithm** that can mechanically construct the specific network $f_{\boldsymbol{\theta}}^{(n)}$ on-the-fly, once the input length n is known.

Remark (Difference between Variable-Length Input NN and an Infinite Sequence of NNs)

A crucial point is that we don't need to store an infinite number of networks $f_{\boldsymbol{\theta}}^{(n)}$ for all possible lengths n.

Instead, we only need an **algorithm** that can mechanically construct the specific network $f_{\boldsymbol{\theta}}^{(n)}$ on-the-fly, once the input length n is known.

This algorithm takes the input length, n, as an argument and dynamically constructs the appropriate network architecture for that specific length. All we need to store permanently is the fixed, finite-dimensional parameter vector $\boldsymbol{\theta}$.

3.2.2 Handling Probability Constraints

How do we ensure the output vector represents a probability mass function?

3.2.2 Handling Probability Constraints

How do we ensure the output vector represents a probability mass function?

The standard method is **softmax normalization**.

Definition (Construction via Softmax Normalization)

Given a variable-length input function $f_{\boldsymbol{\theta}}^{(\cdot)}$ that maps a sequence of length n to a vector of scores in $\mathbb{R}^{|\mathcal{V}|}$, $f_{\boldsymbol{\theta}}^{(n)} : \mathcal{V}^n \to \mathbb{R}^{|\mathcal{V}|}$.

3.2.2 Handling Probability Constraints

How do we ensure the output vector represents a probability mass function?

The standard method is **softmax normalization**.

Definition (Construction via Softmax Normalization)

Given a variable-length input function $f_{\boldsymbol{\theta}}^{(\cdot)}$ that maps a sequence of length n to a vector of scores in $\mathbb{R}^{|\mathcal{V}|}$, $f_{\boldsymbol{\theta}}^{(n)} : \mathcal{V}^n \to \mathbb{R}^{|\mathcal{V}|}$.

The probability of token v given history $\boldsymbol{t}$ is defined as:

$$P_{f_{\boldsymbol{\theta}}^{(\cdot)}}(v \mid \boldsymbol{t}) := \frac{\exp\left(f_{\boldsymbol{\theta},v}^{(|\boldsymbol{t}|)}(\boldsymbol{t})\right)}{\sum_{u \in \mathcal{V}} \exp\left(f_{\boldsymbol{\theta},u}^{(|\boldsymbol{t}|)}(\boldsymbol{t})\right)} \tag{2}$$

The resulting function $P_{f_{\boldsymbol{\theta}}^{(\cdot)}}(\cdot \mid \cdot)$ is a probabilistic language model.

3.2.2 Handling Probability Constraints

In practice, we often introduce a hyperparameter called **temperature** ($T > 0$) to control the shape of the probability distribution.

Definition (Construction with Temperature)

The probability is defined by dividing the scores by T before applying softmax:

$$P_{f_{\boldsymbol{\theta}}^{(\cdot)},T}(v \mid \boldsymbol{t}) := \frac{\exp\left(\frac{1}{T} \cdot f_{\boldsymbol{\theta},v}^{(|\boldsymbol{t}|)}(\boldsymbol{t})\right)}{\sum_{u \in \mathcal{V}} \exp\left(\frac{1}{T} \cdot f_{\boldsymbol{\theta},u}^{(|\boldsymbol{t}|)}(\boldsymbol{t})\right)} \tag{3}$$

Remark

Changing the temperature T [2] controls the "peakedness" of the distribution.

- As $T \downarrow 0$ (low temperature): The distribution becomes sharper. Differences in scores are exaggerated, making the model more confident and deterministic. The probability mass concentrates on the highest-scoring token.

[2] 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).

Remark

Changing the temperature T [2] controls the "peakedness" of the distribution.

- As $T \downarrow 0$ (low temperature): The distribution becomes sharper. Differences in scores are exaggerated, making the model more confident and deterministic. The probability mass concentrates on the highest-scoring token.
- As $T \uparrow \infty$ (high temperature): The distribution becomes flatter, approaching a uniform distribution. The model becomes more random and diverse, giving even low-scoring tokens a chance.

[2] 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).

Remark

Changing the temperature T [2] controls the "peakedness" of the distribution.

- As $T \downarrow 0$ (low temperature): The distribution becomes sharper. Differences in scores are exaggerated, making the model more confident and deterministic. The probability mass concentrates on the highest-scoring token.
- As $T \uparrow \infty$ (high temperature): The distribution becomes flatter, approaching a uniform distribution. The model becomes more random and diverse, giving even low-scoring tokens a chance.
- When $T = 1$, we recover the original softmax definition.

[2] 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).

Token Generators

4.1 Two Basic Ideas

A probabilistic language model $P(\cdot \mid \boldsymbol{t})$ gives us the probability for the <u>next</u> token.
How do we use this to generate an entire sequence?

A probabilistic language model $P(\cdot \mid \boldsymbol{t})$ gives us the probability for the next token. How do we use this to generate an entire sequence?

There are two fundamental strategies.

- **Sampling (Stochastic):** At each step, randomly draw the next token according to the probability distribution $P(\cdot \mid \boldsymbol{t})$. This introduces diversity.

4.1 Two Basic Ideas

A probabilistic language model $P(\cdot \mid \boldsymbol{t})$ gives us the probability for the next token. How do we use this to generate an entire sequence?

There are two fundamental strategies.

- **Sampling (Stochastic):** At each step, randomly draw the next token according to the probability distribution $P(\cdot \mid \boldsymbol{t})$. This introduces diversity.
- **Selecting the Maximum (Deterministic):** At each step, simply pick the token with the highest probability. This leads to more consistent but potentially repetitive outputs.

Sampling requires a source of randomness.

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

Sampling requires a source of randomness.

- True random numbers are hard to obtain on computers.

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

4.1 Two Basic Ideas

Sampling requires a source of randomness.

- True random numbers are hard to obtain on computers.
- We use a **pseudo-random number generator** (PRNG).

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

4.1 Two Basic Ideas

Sampling requires a source of randomness.

- True random numbers are hard to obtain on computers.
- We use a **pseudo-random number generator** (PRNG).
- A PRNG uses a **deterministic algorithm** and an initial **seed** to produce a sequence of numbers that appears random (long period, high uniformity).

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

4.1 Two Basic Ideas

Sampling requires a source of randomness.

- True random numbers are hard to obtain on computers.
- We use a **pseudo-random number generator** (PRNG).
- A PRNG uses a **deterministic algorithm** and an initial **seed** to produce a sequence of numbers that appears random (long period, high uniformity).
- A famous example is the **Mersenne Twister**[3] [3].

---

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

4.2 Definition: Pseudo-random Sequences and Token Generators

First, let's define a pseudo-random sequence.

Definition (Pseudo-random sequence)

Given a fixed initial seed $s \in \mathbb{Z}$, a generator PRNG deterministically returns an infinite sequence of numbers in $[0, 1)$, $\boldsymbol{u}(s) = (u_1, u_2, \dots)$.

4.2 Definition: Pseudo-random Sequences and Token Generators

First, let's define a pseudo-random sequence.

Definition (Pseudo-random sequence)

Given a fixed initial seed $s \in \mathbb{Z}$, a generator PRNG deterministically returns an infinite sequence of numbers in $[0, 1)$, $\boldsymbol{u}(s) = (u_1, u_2, \dots)$.

Now, we can define a token generator.

Definition (Token Generator (General Form))

A **token generator** is a deterministic function $\mathsf{Gen}_{(P,\,\mathsf{PRNG})}$ that takes an input sequence $\boldsymbol{t}_{\text{in}}$ and a seed s, and returns an output sequence $\boldsymbol{t}_{\text{out}}$.

$$\mathsf{Gen}_{(P,\,\mathsf{PRNG})} : \mathcal{V}^* \times \mathbb{Z} \to \mathcal{V}^* \tag{4}$$

Internally, it sequentially appends tokens based on the model P and the numbers from $\boldsymbol{u}(s)$, stopping when a **termination condition** is met.

Remark

If the initial seed s is fixed, the output is **completely deterministic**.

4.2 Definition: Pseudo-random Sequences and Token Generators

Remark

If the initial seed s is fixed, the output is **completely deterministic**.

The apparent "randomness" comes entirely from the PRNG. The generation rule itself is a deterministic mathematical function.

4.3 Sampling

This is the most basic sampling method, also known as inverse transform sampling.

Definition (Sampling using a Probabilistic Language Model)

Given an initial sequence $\boldsymbol{t}^{(0)}$, a probabilistic model P, a PRNG, and a stopping condition Stop. For $i = 0, 1, 2, \ldots$:

1. If $\mathsf{Stop}(\boldsymbol{t}^{(i)})$ is true, stop and return $\boldsymbol{t}^{(i)}$.

4.3 Sampling

This is the most basic sampling method, also known as inverse transform sampling.

Definition (Sampling using a Probabilistic Language Model)

Given an initial sequence $\boldsymbol{t}^{(0)}$, a probabilistic model P, a PRNG, and a stopping condition Stop. For $i = 0, 1, 2, \ldots$:

1. If $\mathsf{Stop}(\boldsymbol{t}^{(i)})$ is true, stop and return $\boldsymbol{t}^{(i)}$.
2. Calculate the cumulative distribution function (CDF): $F^{(i)}(v) := \sum_{u \le v} P\left(u | \boldsymbol{t}^{(i)}\right)$.

4.3 Sampling

This is the most basic sampling method, also known as inverse transform sampling.

Definition (Sampling using a Probabilistic Language Model)

Given an initial sequence $\boldsymbol{t}^{(0)}$, a probabilistic model P, a PRNG, and a stopping condition Stop. For $i = 0, 1, 2, \ldots$:

1. If $\mathsf{Stop}(\boldsymbol{t}^{(i)})$ is true, stop and return $\boldsymbol{t}^{(i)}$.
2. Calculate the cumulative distribution function (CDF): $F^{(i)}(v) := \sum_{u \leq v} P\left(u|\boldsymbol{t}^{(i)}\right)$.
3. Get the next random number u_{i+1} from the PRNG.

This is the most basic sampling method, also known as inverse transform sampling.

Definition (Sampling using a Probabilistic Language Model)

Given an initial sequence $\boldsymbol{t}^{(0)}$, a probabilistic model P, a PRNG, and a stopping condition Stop. For $i = 0, 1, 2, \ldots$:

1. If $\mathsf{Stop}(\boldsymbol{t}^{(i)})$ is true, stop and return $\boldsymbol{t}^{(i)}$.
2. Calculate the cumulative distribution function (CDF): $F^{(i)}(v) := \sum_{u \leq v} P\left(u | \boldsymbol{t}^{(i)}\right)$.
3. Get the next random number u_{i+1} from the PRNG.
4. Find the next token: $t_{i+1} := \min\{\, v \in \mathcal{V} \mid F^{(i)}(v) > \; u_{i+1} \,\}$.

4.3 Sampling

This is the most basic sampling method, also known as inverse transform sampling.

Definition (Sampling using a Probabilistic Language Model)

Given an initial sequence $\boldsymbol{t}^{(0)}$, a probabilistic model P, a PRNG, and a stopping condition Stop. For $i = 0, 1, 2, \ldots$:

1. If $\mathsf{Stop}(\boldsymbol{t}^{(i)})$ is true, stop and return $\boldsymbol{t}^{(i)}$.
2. Calculate the cumulative distribution function (CDF): $F^{(i)}(v) \coloneqq \sum_{u \leq v} P\left(u | \boldsymbol{t}^{(i)}\right)$.
3. Get the next random number u_{i+1} from the PRNG.
4. Find the next token: $t_{i+1} \coloneqq \min\{\, v \in \mathcal{V} \mid F^{(i)}(v) > \, u_{i+1} \,\}$.
5. Append the token: $\boldsymbol{t}^{(i+1)} \coloneqq (\boldsymbol{t}^{(i)}, \, t_{i+1})$ and repeat.

Example (Two Steps of Naive Sampling)

Let $\mathcal{V} = \{1, 2, 3\}$, input $\boldsymbol{t}_{\text{in}} = ()$, and stop at length 2. The probability distributions are:

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

The PRNG gives us $u_1 = 0.68,\ u_2 = 0.72$. Let's find the output.

4.3 Sampling

Step 1:

- Current sequence $\boldsymbol{t}^{(0)} = ()$.
- The probability distribution is $P(\cdot|()) = (0.50,\ 0.30,\ 0.20)$.
- The cumulative distribution (CDF) is $F^{(0)} = (0.50, 0.80, 1.00)$.
- Our first random number is $u_1 = 0.68$.
- We look for the smallest token v where $F^{(0)}(v) > 0.68$.
  - $F^{(0)}(1) = 0.50 \not> 0.68$
  - $F^{(0)}(2) = 0.80 > 0.68$
- So, the next token is $t_1 = 2$. New sequence $\boldsymbol{t}^{(1)} = (2)$.

4.3 Sampling

Step 2:

- Current sequence $\boldsymbol{t}^{(1)} = (2)$.
- The probability distribution is $P(\cdot|(2)) = (0.10,\ 0.20,\ 0.70)$.
- The CDF is $F^{(1)} = (0.10, 0.30, 1.00)$.
- Our second random number is $u_2 = 0.72$.
- We look for the smallest token v where $F^{(1)}(v) > 0.72$.
  - $F^{(1)}(1) = 0.10 \not> 0.72$
  - $F^{(1)}(2) = 0.30 \not> 0.72$
  - $F^{(1)}(3) = 1.00 > 0.72$
- So, the next token is $t_2 = 3$. New sequence $\boldsymbol{t}^{(2)} = (2, 3)$.

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

Exercise (Manual Calculation Exercise (Naive Sampling))

Let $\mathcal{V} = \{1, 2, 3\}$, start with $()$, and stop at length 2.

$$P(\cdot \mid ()) = (0.40,\ 0.40,\ 0.20)$$
$$P(\cdot \mid (1)) = (0.25,\ 0.25,\ 0.50)$$
$$P(\cdot \mid (2)) = (0.10,\ 0.70,\ 0.20)$$
$$P(\cdot \mid (3)) = (0.50,\ 0.30,\ 0.20)$$

Given $u_1 = 0.41,\ u_2 = 0.20$, find the output sequence.

Answer

Step 1:

- $\boldsymbol{t}^{(0)} = ()$.
- $P(\cdot|()) = (0.40,\ 0.40,\ 0.20)$.
- CDF: $(0.40, 0.80, 1.00)$.
- $u_1 = 0.41$. Since $0.40 < 0.41 \leq 0.80$, we select $t_1 = 2$.
- New sequence: $\boldsymbol{t}^{(1)} = (2)$.

Answer

Step 1:

- $\boldsymbol{t}^{(0)} = ()$.
- $P(\cdot|()) = (0.40,\ 0.40,\ 0.20)$.
- CDF: $(0.40, 0.80, 1.00)$.
- $u_1 = 0.41$. Since $0.40 < 0.41 \leq 0.80$, we select $t_1 = 2$.
- New sequence: $\boldsymbol{t}^{(1)} = (2)$.

Step 2:

- $\boldsymbol{t}^{(1)} = (2)$.
- $P(\cdot|(2)) = (0.10,\ 0.70,\ 0.20)$.
- CDF: $(0.10, 0.80, 1.00)$.

4.4 Modifying Probabilistic Language Models

In practice, the raw probability distribution from the model is often modified before sampling.

4.4 Modifying Probabilistic Language Models

In practice, the raw probability distribution from the model is often modified before sampling.

Motivations:

- **Preventing Repetition:** Avoid generating monotonous, repetitive text.

4.4 Modifying Probabilistic Language Models

In practice, the raw probability distribution from the model is often modified before sampling.

Motivations:

- **Preventing Repetition:** Avoid generating monotonous, repetitive text.
- **Excluding Low-Score Tokens:** Avoid generating nonsensical text by completely ruling out tokens that the model considers very unlikely.

Here are some popular techniques:

- **Repetition Penalty [2]:** Decrease the scores of tokens that have already appeared in the generated sequence.

4.4 Modifying Probabilistic Language Models

Here are some popular techniques:

- **Repetition Penalty [2]:** Decrease the scores of tokens that have already appeared in the generated sequence.
- **Temperature:** As we've seen, lowering $T < 1$ makes the distribution sharper and less random.

Here are some popular techniques:

- **Repetition Penalty [2]:** Decrease the scores of tokens that have already appeared in the generated sequence.
- **Temperature:** As we've seen, lowering $T < 1$ makes the distribution sharper and less random.
- **Top-k Sampling:** Consider only the top k most probable tokens and set the probability of all others to zero.

4.4 Modifying Probabilistic Language Models

Here are some popular techniques:

- **Repetition Penalty [2]:** Decrease the scores of tokens that have already appeared in the generated sequence.
- **Temperature:** As we've seen, lowering $T < 1$ makes the distribution sharper and less random.
- **Top-k Sampling:** Consider only the top k most probable tokens and set the probability of all others to zero.
- **Top-p (Nucleus) Sampling [1]:** Consider the smallest set of most probable tokens whose cumulative probability exceeds a threshold p. This set is called the "nucleus."

These modifications are often applied in a specific order. Let's look at the common pipeline used by libraries like Hugging Face.

Definition (Construction of Conditional PMF in Hugging Face)

Given scores $\boldsymbol{s}_{|\boldsymbol{t}}$ for history $\boldsymbol{t}$:

1. Apply **repetition penalty**.
2. Apply **temperature** scaling.
3. Apply **top-k** filtering.
4. Apply **softmax** to get probabilities $\pi(v)$.
5. Apply **top-p** (nucleus) filtering and renormalize.

Let's look closer at the repetition penalty step.

1. (**repetition penalty**) Let $H(\boldsymbol{t})$ be the set of previously seen tokens and $\lambda > 0$ be the penalty factor.

$$s^{\mathrm{rep}}_{v|\boldsymbol{t}} \leftarrow \begin{cases} s_{v|\boldsymbol{t}} & \text{if } v \notin H(\boldsymbol{t}), \\ \frac{1}{\lambda} s_{v|\boldsymbol{t}} & \text{if } v \in H(\boldsymbol{t}) \text{ and } s_{v|\boldsymbol{t}} \geq 0, \\ \lambda s_{v|\boldsymbol{t}} & \text{if } v \in H(\boldsymbol{t}) \text{ and } s_{v|\boldsymbol{t}} < 0. \end{cases}$$

Example (Hugging Face Style Sampling)

Let $\mathcal{V} = \{1, 2, 3, 4\}$, start with $()$, and stop at length 2. Scores:

$$\boldsymbol{s}_{|()} = (2.0,\ 1.0,\ 0.0,\ -1.0)$$
$$\boldsymbol{s}_{|(1)} = (1.5,\ 0.0,\ 0.5,\ -0.2)$$

Hyperparameters: $T = 0.5,\ \lambda = 1.2,\ k = 3,\ p = 0.9$. PRNG gives $u_1 = 0.50, u_2 = 0.90$. Find the output.

Step 1 (History $\boldsymbol{t} = ()$, Seen tokens $H(()) = \{\}$)

1. **(rep)** No tokens seen, scores unchanged: $(2.0, 1.0, 0.0, -1.0)$.
2. **(temp)** Divide by $T = 0.5$ (i.e., multiply by 2): $(4.0, 2.0, 0.0, -2.0)$.
3. **(top-k)** Keep top $k = 3$ scores. 4th score becomes $-\infty$: $(4.0, 2.0, 0.0, -\infty)$.
4. **(softmax)** Probabilities $\pi \approx (0.8668, 0.1173, 0.0159, 0)$.
5. **(top-p)** Cumulative sum is $(0.8668, 0.9841, \dots)$. For $p = 0.9$, nucleus is $\{1, 2\}$. Renormalize: $\tilde{\pi} \approx (0.8808, 0.1192, 0, 0)$.
6. **(sample)** CDF is $(0.8808, 1.0)$. $u_1 = 0.50 < 0.8808 \implies t_1 = 1$.

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

1. **(rep)** Score for token 1: $1.5/\lambda = 1.5/1.2 = 1.25$. Scores: $(1.25, 0.0, 0.5, -0.2)$.
2. **(temp)** Divide by $T = 0.5$: $(2.5, 0.0, 1.0, -0.4)$.
3. **(top-k)** Keep top $k = 3$ scores: $(2.5, 0.0, 1.0, -\infty)$.
4. **(softmax)** Probabilities $\pi \approx (0.7662, 0.0629, 0.1710, 0)$.
5. **(top-p)** Sorted probs: $(0.7662_1, 0.1710_3, 0.0629_2)$. Cum sum: $(0.7662, 0.9372, \dots)$. Nucleus is $\{1, 3\}$. Renormalize: $\tilde{\pi} \approx (0.8176, 0, 0.1824, 0)$.
6. **(sample)** CDF is $(0.8176, 0.8176, 1.0, 1.0)$. $u_2 = 0.90$. Since $0.8176 < 0.90 \leq 1.0$, we select $t_2 = 3$.

Final output: $(1, 3)$.

4.5 Selecting the Maximum Probability Element

Let's now turn to the deterministic approach. The simplest method is greedy search.

Definition (Greedy decoding)

Given an initial sequence $\boldsymbol{t}^{(0)}$ and a stopping condition Stop. At each step i:

1. Select the token with the highest probability:

$$t_{i+1} := \arg\max_{v \in \mathcal{V}} P\left(v \middle| \boldsymbol{t}^{(i)}\right)$$

   (Ties are broken by a fixed rule, e.g., lowest token ID).
2. Append the token: $\boldsymbol{t}^{(i+1)} := (\boldsymbol{t}^{(i)}, t_{i+1})$.
3. Repeat until Stop is met.

Example (Greedy Search (Two Steps, Numerical))

Let $\mathcal{V} = \{1, 2, 3\}$, stop at length 2. Probabilities:

$$P(\cdot \mid ()) = (0.55,\ 0.40,\ 0.05)$$
$$P(\cdot \mid (1)) = (0.20,\ 0.70,\ 0.10)$$

- **Step 1:** Given $()$, the probabilities are $(0.55, 0.40, 0.05)$. The max is at index 1. So, $t_1 = 1$.
- **Step 2:** Given (1), the probabilities are $(0.20, 0.70, 0.10)$. The max is at index 2. So, $t_2 = 2$.

The output is $(1, 2)$.

4.6 Limitations of Greedy Search and Beam Search

Greedy search does not guarantee finding the sequence with the highest overall joint probability.

4.6 Limitations of Greedy Search and Beam Search

Greedy search does not guarantee finding the sequence with the highest overall joint probability.

The joint probability of a sequence is $P(t_1, t_2) = P(t_1 \mid ()) \cdot P(t_2 \mid (t_1))$. A locally optimal choice at step 1 might lead to a globally suboptimal sequence.

Example (Greedy Fails)

Let $\mathcal{V} = \{A, B\}$, stop at length 2.

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

4.6 Limitations of Greedy Search and Beam Search

Greedy Search Path:

- Step 1: $P(A|()) = 0.6 > P(B|()) = 0.4$. Choose A.
- Step 2: $P(A|(A)) = 0.5$. Choose A (tie-break).
- Result: (A, A). Joint probability: $P(A, A) = 0.6 \times 0.5 = 0.30$.

4.6 Limitations of Greedy Search and Beam Search

Greedy Search Path:

- Step 1: $P(A|()) = 0.6 > P(B|()) = 0.4$. Choose A.
- Step 2: $P(A|(A)) = 0.5$. Choose A (tie-break).
- Result: (A, A). Joint probability: $P(A, A) = 0.6 \times 0.5 = 0.30$.

True Best Sequence: Let's check all possibilities:

- $P(A, A) = 0.6 \times 0.5 = 0.30$
- $P(A, B) = 0.6 \times 0.5 = 0.30$
- $P(B, A) = 0.4 \times 0.95 = \mathbf{0.38}$
- $P(B, B) = 0.4 \times 0.05 = 0.02$

The sequence (B, A) has the highest probability, but greedy search missed it.

4.6 Limitations of Greedy Search and Beam Search

Beam search is an improvement over greedy search that addresses this issue.

4.6 Limitations of Greedy Search and Beam Search

Beam search is an improvement over greedy search that addresses this issue.

Idea: Instead of keeping only the single best choice at each step, keep track of the B most probable partial sequences (the "beam").

Definition (Beam Search (Beam width B))

1. Start with an empty sequence.
2. At each step i:
   - For each of the B sequences currently in the beam, generate all $|\mathcal{V}|$ possible next sequences and calculate their log-probabilities.
   - From this set of $B \times |\mathcal{V}|$ candidates, select the top B sequences with the highest overall log-probabilities. These form the new beam.
3. Repeat until a stop condition is met, then return the highest-scoring sequence from the final beam.

4.6 Limitations of Greedy Search and Beam Search

Let’s re-run the previous example with beam search, using a beam width $B = 2$.

Step 1:

- Initial candidates: (A) with log-prob $\log(0.6)$ and (B) with log-prob $\log(0.4)$.
- Since $B = 2$, we keep both.
- Beam $\mathcal{B}_1 = \{(A, \log 0.6), (B, \log 0.4)\}$.

4.6 Limitations of Greedy Search and Beam Search

Let's re-run the previous example with beam search, using a beam width $B = 2$.

Step 1:

- Initial candidates: (A) with log-prob $\log(0.6)$ and (B) with log-prob $\log(0.4)$.
- Since $B = 2$, we keep both.
- Beam $\mathcal{B}_1 = \{(A, \log 0.6), (B, \log 0.4)\}$.

Step 2:

- Expand from A: (A, A) with log-prob $\log(0.6) + \log(0.5) = \log(0.3)$. And (A, B) also with log-prob $\log(0.3)$.
- Expand from B: (B, A) with log-prob $\log(0.4) + \log(0.95) = \log(0.38)$. And (B, B) with log-prob $\log(0.02)$.
- Candidates' log-probs: $\{\log 0.30, \log 0.30, \log 0.38, \log 0.02\}$.
- The top 2 are (B, A) and (A, A) (or (A, B)). The best is (B, A).

Summary

5. Summary

Let's summarize the key takeaways from today's lecture.

- Neural networks, being continuous functions, are naturally used as **probabilistic language models**. They output scores, which are converted to a probability mass function (PMF) via softmax. The joint probability of a sequence is then defined by the chain rule.

5. Summary

Let's summarize the key takeaways from today's lecture.

- Neural networks, being continuous functions, are naturally used as **probabilistic language models**. They output scores, which are converted to a probability mass function (PMF) via softmax. The joint probability of a sequence is then defined by the chain rule.
- We can rigorously define various text generation methods within a unified framework of a **token generator**, which is a deterministic function of a model, a stopping condition, and a seed for a pseudo-random number generator.

5. Summary

Let's summarize the key takeaways from today's lecture.

- Neural networks, being continuous functions, are naturally used as **probabilistic language models**. They output scores, which are converted to a probability mass function (PMF) via softmax. The joint probability of a sequence is then defined by the chain rule.
- We can rigorously define various text generation methods within a unified framework of a **token generator**, which is a deterministic function of a model, a stopping condition, and a seed for a pseudo-random number generator.
- This framework includes methods like greedy search, beam search, and various sampling techniques that modify the probability distribution using temperature, top-k, top-p, and repetition penalty.

Next Time

7. Next Time

Next time, we will discuss the **embedding layer**.

7. Next Time

Next time, we will discuss the **embedding layer**.

This is the initial layer in many probabilistic language models. We will focus on:

- How it maps discrete tokens from the vocabulary into a continuous vector space.

7. Next Time

Next time, we will discuss the **embedding layer**.

This is the initial layer in many probabilistic language models. We will focus on:

- How it maps discrete tokens from the vocabulary into a continuous vector space.
- The learning rules for this mapping.

7. Next Time

Next time, we will discuss the **embedding layer**.

This is the initial layer in many probabilistic language models. We will focus on:

- How it maps discrete tokens from the vocabulary into a continuous vector space.
- The learning rules for this mapping.
- Its statistical interpretation.

References i

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

[2] 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.

[3] 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.

[4] 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.