# Unit 08 - Language Modeling (Extended Technical Article)

Arya Sanjay - Interactive Intelligence

## 1 What is Language Modeling?

Language Modeling (LM) is about building systems that can understand, generate, or transform human language. At its simplest, a language model predicts the next word in a sequence based on the words that came before it. This ability forms the foundation of many applications in **natural language processing (NLP)**, such as chatbots (e.g., ChatGPT), machine translation, text summarization, and search engines.
Let's consider an example:

*"The quick brown fox jumps ___."*

Given this sentence, a language model would likely predict the word "over" as the next word, drawing on patterns it has learned from analyzing text. By recognizing and using patterns in language, these models can:

- Complete sentences.
- Understand context.
- Generate coherent responses or paragraphs.

These are examples of text-generation tasks, where language models predict or create text based on input. Language models also support other types of NLP tasks, such as classification or summarization.
Language models work by estimating the probability of a sequence of words. Formally, the model breaks this problem into smaller steps, predicting one word at a time based on the words before it:

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_n|w_1, w_2, \ldots, w_{n-1})$$

This equation may look complex, but it simply means that the probability of an entire sequence is determined by multiplying the probabilities of each word, one after another, given their context.

### 1.1 How Are Language Models Trained?

To train a language model, we show it large amounts of text and teach it to minimize its prediction errors. Specifically, we compare the words it predicts with the actual next words in the text and adjust the model to get better over time. The measure of prediction error is called **cross-entropy loss**:

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{T_i}\log P(w_j^{(i)}|w_1^{(i)},\ldots,w_{j-1}^{(i)})$$

While this might seem technical, the idea is simple: the closer the model's predictions are to the actual text, the smaller this loss becomes. Training involves minimizing this loss so the model becomes better at predicting language patterns.

### 1.2 What You'll Learn in This Article

This article will guide you through the essentials of language modeling, including:

- **Core Concepts** like tokenization, embeddings, and sequence modeling.
- **Architectures** such as Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and Transformers.
- **Applications** in areas like chatbots, translation, and sentiment analysis.
- **Challenges** like managing bias, hallucinations, and the scale of large models.

By the end, you'll understand the theoretical foundations of language models, how they're built and trained, and their real-world impact.

**Synthesis Questions:**

1. Why is it important for language models to predict the likelihood of sequences accurately? Give a practical example where this capability is crucial.
2. How do language models like GPT differ in their approach to understanding language compared to traditional rule-based systems?

## 2 Foundational Concepts

### 2.1 Tokenization: The First Step

Tokenization breaks raw text into smaller units called **tokens**, which are the building blocks for language models. Tokenization is essential because language

models cannot directly process text; they operate on numerical data. The granularity of tokens varies:

- **Word-level Tokenization:** Splits text into individual words or phrases.
- **Character-level Tokenization:** Breaks text into single characters, useful for languages with complex morphologies.
- **Subword Tokenization:** Balances granularity and vocabulary efficiency, often employed in modern models like BERT and GPT.

**Examples of Tokenization:**

- Word-level:

```
Sentence: "Arya is amazing!"
Tokens: ["Arya", "is", "amazing", "!"]
```

- Character-level:

```
Sentence: "Arya"
Tokens: ["A", "r", "y", "a"]
```

- Subword-level:

```
Sentence: "unbelievable"
Tokens: ["un", "believable"]
```

Algorithms like Byte Pair Encoding (BPE) and WordPiece generate subword tokens, enabling the model to handle unseen words by breaking them into meaningful subunits. For example, "transformational" can be tokenized as ["transform", "ational"], capturing semantic and morphological relationships.

**Synthesis Questions:**

1. Compare and contrast word-level, character-level, and subword tokenization. In which scenarios might each be most appropriate?
2. What are the advantages of using subword tokenization for handling out-of-vocabulary words? Provide an example.
3. Given a language with agglutinative morphology (e.g., Turkish), which tokenization approach would likely work best and why?

### 2.2 Embeddings: Representing Tokens Numerically

After tokenization, tokens are represented as discrete symbols, which must be converted into numerical vectors for processing by language models. **Embeddings** achieve this transformation by mapping tokens to dense vector spaces where semantic and syntactic relationships are preserved. These embeddings play a crucial role in enabling models to understand and process natural language effectively. Embeddings are typically **dense vectors**, meaning they are compact numerical representations with most elements being non-zero. This property allows embeddings to efficiently capture semantic and syntactic relationships between tokens compared to sparse vectors, which contain many zeros.

**Why Embeddings Matter:** Embeddings play a crucial role in bridging the gap between human language and machine computation, enabling efficient downstream processing.

**Key Properties of Embeddings:**

- **Semantic Similarity:** Tokens with similar meanings have similar embeddings, enabling the model to capture linguistic relationships. Cosine similarity is preferred as it focuses on the angular distance between vectors, ignoring magnitude differences that may arise due to varying token frequencies.
- **Contextual Adaptability:** Contextual embeddings dynamically adjust based on sentence context, allowing the model to handle polysemous words (e.g., "bank" can refer to a financial institution or a riverbank).

**Mathematical Representation:**

$$\text{Embedding(token)} = \mathbf{e} \in \mathbb{R}^d$$

where $d$ is the dimensionality of the embedding space. Each token is represented as a point in this $d$-dimensional space, capturing its relationships with other tokens.

**Types of Embeddings:**

- **Pre-trained Static Embeddings:**
  - **Word2Vec:** Learns embeddings by maximizing the cosine similarity between words appearing in similar contexts.
  - **GloVe (Global Vectors):** Embeds words by factorizing a co-occurrence matrix to capture statistical properties of word distributions.
  - **FastText:** Enhances embeddings by incorporating subword information, improving robustness for rare and out-of-vocabulary words.

- **Contextual Embeddings:** Advanced models like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformers) generate embeddings that depend on the surrounding context of a token. For example:

  ```
  Sentence: "The bank is on the riverbank."
  ```

  The embedding for the word "bank" differs in the financial and river contexts, reflecting its contextual meaning.

- **Embedding Arithmetic:** A unique property of embeddings is their ability to encode semantic relationships through arithmetic operations. For instance:

  $$\text{Embedding(king)} - \text{Embedding(man)} + \text{Embedding(woman)} \approx \text{Embedding(queen)}$$

  This illustrates how embeddings capture relationships between words in dense vector spaces.

**Synthesis Questions:**

1. Explain the significance of dense vector spaces in embeddings. Why is cosine similarity often used to measure relationships between embeddings?
2. How do contextual embeddings differ from static embeddings? Illustrate with an example involving polysemy.
3. How would you adapt static embeddings for a multilingual task? What challenges arise?
4. Suppose you need to train embeddings on a highly domain-specific dataset (e.g., medical texts). What modifications or strategies might you employ to make embeddings effective in this context?

### 2.3 Sequence Modeling: The Core Objective

At the heart of language modeling lies the task of sequence prediction, which involves estimating the probability of a sequence of tokens. Autoregressive models, such as GPT, predict the next token in a sequence by conditioning on all previous tokens. This process is iterative, generating one token at a time, which allows these models to produce coherent sequences of text.
Mathematically, this can be expressed as:

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{<t})$$

Here:

- $x_t$ represents the current token at time step $t$.
- $x_{<t}$ denotes all tokens preceding $x_t$ (i.e., the context or history).

**Elaborative Example:** Let us break down the sequence $P(\text{"The quick brown fox jumps"})$ step by step using the chain rule of probability. Here, the sequence contains five tokens: "The", "quick", "brown", "fox", and "jumps". The joint probability of the sequence is decomposed as:

$$P(\text{"The quick brown fox jumps"}) = P(\text{"The"}) \cdot P(\text{"quick"} \mid \text{"The"}) \cdot P(\text{"brown"} \mid \text{"The quick"}) \cdot P(\text{"fox"} \mid \text{"The quick brown"}) \cdot P(\text{"jumps"} \mid \text{"The quick brown fox"})$$

Each term in this product represents the conditional probability of a token given the tokens that precede it in the sequence.

**Step-by-Step Calculation (Hypothetical Probabilities):**

- $P(\text{"The"}) = 0.4$
- $P(\text{"quick"} \mid \text{"The"}) = 0.3$
- $P(\text{"brown"} \mid \text{"The quick"}) = 0.2$
- $P(\text{"fox"} \mid \text{"The quick brown"}) = 0.5$
- $P(\text{"jumps"} \mid \text{"The quick brown fox"}) = 0.6$

Thus, the joint probability of the entire sequence is calculated as:

$$\begin{aligned} P(\text{"The quick brown fox jumps"}) &= 0.4 \cdot 0.3 \cdot 0.2 \cdot 0.5 \cdot 0.6 \\ &= 0.0072 \end{aligned}$$

### 2.4 Sequence Modeling: The Core Objective

At the heart of language modeling lies the task of sequence prediction, which involves estimating the probability of a sequence of tokens. Meanwhile, autoregressive models, such as GPT, predict the next token in a sequence by conditioning on all previous tokens. This process is iterative, generating one token at a time, which allows these models to produce coherent sequences of text. Sequence prediction can be mathematically expressed as:

$$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t | x_{<t})$$

Here:

- $x_t$ represents the current token at time step $t$.
- $x_{<t}$ denotes all tokens preceding $x_t$ (i.e., the context or history).

**Elaborative Example:**
Let us break down the sequence $P(\text{"The quick brown fox jumps"})$ step by step using the chain rule of probability. Here, the sequence contains five tokens: "The", "quick", "brown", "fox", and "jumps". The joint probability of the sequence is decomposed as:

$$\begin{aligned} P(\text{"The quick brown fox jumps"}) = {} & P(\text{"The"}) \cdot P(\text{"quick"} \mid \text{"The"}) \\ & \cdot P(\text{"brown"} \mid \text{"The quick"}) \\ & \cdot P(\text{"fox"} \mid \text{"The quick brown"}) \\ & \cdot P(\text{"jumps"} \mid \text{"The quick brown fox"}) \end{aligned}$$

Each term in this product represents the conditional probability of a token given the tokens that precede it in the sequence.
**Step-by-Step Calculation (Hypothetical Probabilities):**

- $P(\text{"The"}) = 0.4$
- $P(\text{"quick"} \mid \text{"The"}) = 0.3$
- $P(\text{"brown"} \mid \text{"The quick"}) = 0.2$
- $P(\text{"fox"} \mid \text{"The quick brown"}) = 0.5$
- $P(\text{"jumps"} \mid \text{"The quick brown fox"}) = 0.6$

Thus, the joint probability of the entire sequence is calculated as:

$$P(\text{"The quick brown fox jumps"}) = 0.4 \cdot 0.3 \cdot 0.2 \cdot 0.5 \cdot 0.6 = 0.0072$$

This step-by-step breakdown illustrates how language models leverage the chain rule of probability to predict tokens sequentially while accounting for prior context.
**Example with Numerical Probabilities:** Consider a toy vocabulary with three words: "cat", "dog", and "bird". Given $h_t$, the model computes:

$$P(\text{"cat"}|x_{<t}) = \frac{e^{z_{\text{"cat"}}}}{\sum_{w \in \{\text{"cat"},\text{"dog"},\text{"bird"}\}} e^{z_w}}$$

where $z_{\text{"cat"}} = W \cdot h_t[\text{"cat"}]$. This ensures probabilistic coherence for output predictions.
By chaining these probabilities, the model constructs meaningful sequence-level predictions.

**Training Objective: Negative Log-Likelihood (NLL) Loss**
Language models are typically trained using the **negative log-likelihood (NLL)** loss function, defined as:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(x_t | x_{<t})$$

The NLL loss penalizes the model when it assigns low probabilities to the actual tokens in a sequence. Minimizing this loss ensures the model learns to generate sequences with high probability for real-world language patterns.
**Key Insights:**

- The sequence prediction task assumes a left-to-right (or autoregressive) approach for models like GPT or a bidirectional approach for models like BERT.
- Accurate modeling of $P(x_t|x_{<t})$ requires capturing long-range dependencies, grammatical structure, and semantic coherence within the sequence.

**Illustrative Example:** Consider the sentence:

```
"A journey of a thousand miles begins with a single step."
```

During training, the model learns to predict each word sequentially:

$$P(\text{"A"}),\ P(\text{"journey"}|\text{"A"}),\ P(\text{"of"}|\text{"A journey"}), \ldots$$

The cumulative product of these probabilities represents the likelihood of the entire sentence.
Sequence modeling is the foundation for understanding how language models generate coherent and contextually relevant outputs.

**Synthesis Questions:**

1. Derive the negative log-likelihood loss formula from the sequence probability expression.
2. How does the choice of $P(x_t|x_{<t})$ influence the quality of a language model's output? Provide a concrete example.
3. How might bidirectional modeling (as in BERT) handle $P(x_t|x_{<t})$ differently from autoregressive models like GPT? What are the trade-offs?