Language Modeling

Language Modeling

Intuitive Understanding

Language Modeling is about teaching a machine to **understand and generate human language** — by learning the **probability of word sequences**.

Think of it as helping the model answer:

> "Given the words so far, what word (or token) is most likely to come next?"

Example

Suppose you have these two sentences:

1. "The cat sat on the ___."
2. "The cat sat on the airplane."

A **language model** will assign a **higher probability** to:

> "The cat sat on the mat."

because "mat" is a more likely continuation based on how words co-occur in real language.

Formal Definition

A **Language Model (LM)** estimates the **probability distribution** over sequences of words (or tokens):

$P(w_1, w_2, w_3, ..., w_T)$

For practical use, it learns to compute:

$P(w_t|w_1, w_2, ..., w_{t-1})$

This means:

> The probability of the current word wtw_twt given all previous words.

Types of Language Models

1. Statistical (Classical) Language Models

Before deep learning, LMs used counting-based approaches.

- **Unigram model:** assumes all words are independent

  $\rightarrow P(w_1, w_2, ..., w_T) = \prod_i P(w_i)$

- **Bigram model:** depends only on previous one word

  $\rightarrow P(w_t | w_{t-1})$

- **Trigram model:** depends on last two words

  $\rightarrow P(w_t | w_{t-2}, w_{t-1})$

➡️ **Limitation:** Can't handle long-term dependencies well.

---

2. Neural Language Models

Instead of counting, they **learn** patterns using neural networks.

a. **Feedforward Neural LMs**

- Inputs are fixed-size windows of previous words.
- Output: probability distribution over next word.

b. **Recurrent Neural Networks (RNN / LSTM / GRU)**

- Maintain a hidden state capturing past context.
- Can process sequences of any length.

c. **Transformer-based Models (Modern)**

- Use **self-attention** to capture dependencies between all words in a sequence simultaneously.
- Example models:
  - **GPT (Generative Pretrained Transformer)** — autoregressive LM

- **BERT** — masked LM (predict missing words)

How to build a neural language model?

1. **Gather text data** (corpus) — e.g., WikiText, books, scraped text.
2. **Tokenize** — map text → tokens. Use subword (BPE / WordPiece / SentencePiece) for open vocab.
3. **Create dataset** — convert token ids into training sequences (inputs and targets). Use sliding windows or next-token pairs.
4. **Model** — choose architecture: RNN/LSTM/GRU (simple), Transformer (state of the art).
5. **Loss** — Cross-entropy on next-token prediction.
6. **Optimization** — Adam/AdamW, LR schedule (warmup + decay), gradient clipping.
7. **Evaluation** — Perplexity (exp(avg cross-entropy)), and sample quality.
8. **Inference** — greedy, beam, top-k, top-p (nucleus) sampling for generation.
9. **Deploy/Serve** — convert to ONNX / TorchScript or use model-serving infra.

Fixed-Window Neural Language Model

1. Intuitive Idea

A **Fixed-window Neural Language Model (NNLM)** predicts the next word based on a **fixed number of previous words** (just like an *n-gram*),

But instead of counting co-occurrences, it **learns distributed word embeddings** and uses a **neural network** to generalize.

> Think: "Instead of memorizing all possible 3-word combinations, I'll learn continuous representations of words that let me guess likely next words, even for unseen sequences."

---

2. The Setting

Let's say you have a sequence of words:

$$w_1, w_2, w_3, \ldots, w_T$$

We want to estimate:

$$P(w_t | w_{t-n+1}, \ldots, w_{t-1})$$

This is similar to a trigram or 5-gram model — we only look at a *fixed window* of n−1 previous words.

---

3. Mathematically

Let's define:

- V: vocabulary size
- d: embedding dimension
- n: context window size (number of previous words used)

Each word w_i is represented as a one-hot vector $x_i \in \mathbb{R}^V$.

Step 1: Embedding lookup

We learn an embedding matrix $E \in \mathbb{R}^{V \times d}$.

$$e_i = E^T x_i$$

Now we have n−1 embeddings: $e_{t-n+1}, \ldots, e_{t-1}$.

Step 2: Concatenate

$$z = [e_{t-n+1}; e_{t-n+2}; \ldots ; e_{t-1}] \in \mathbb{R}^{(n-1)d}$$

Step 3: Feedforward neural network

We apply a non-linear transformation:

$$h = \tanh(W_1 z + b_1)$$

Then compute scores for all words in vocabulary:

$$o = W_2 h + b_2$$

Step 4: Softmax output

$P(w_t = i|context) = \frac{\exp(o_i)}{\sum_j \exp(o_j)}$

Step 5: Training

Use **cross-entropy loss** to maximize the probability of the correct next word.

🧠 4. What It Learns

- The **embedding matrix E** learns semantic representations of words (similar words get similar vectors).
- The **neural layers** learn to **combine** context information nonlinearly.
- This overcomes data sparsity of n-grams: it can generalize from *similar* contexts.

---

5. Limitations

Problem	Why It Matters
Fixed window	Can't use longer context (e.g., 10+ previous words) without huge parameter growth.
Parameter explosion	Input layer grows linearly with window size × embedding dim.
No sequence memory	Doesn't "remember" beyond window — no notion of sentence history.

That's why **RNNs** and later **Transformers** replaced it — they can handle **variable-length** context and **capture long dependencies**.