Masked Language Models (BERT) — A Complete Academic Manual

Intuition, math, worked examples, diagrams, and exercises

Colors: blue = concepts, green = examples, red = important notes

Contents

1 Overview — What and Why

This manual explains Bidirectional Transformer encoders (BERT family): how they are trained (Masked Language Modeling and Next-Sentence Prediction), why these objectives work, how the masking procedure operates, the role of positional and segment embeddings, architecture components, training practicalities, and worked examples. It is written as a concise academic tutorial: clear definitions, mathematical formulas, intuition boxes, and exercises.

2 High-level intuition

- **Encoder focus:** BERT-style models are *encoders* that produce contextual token vectors by looking both left and right.
- **MLM (denoising):** training corrupts input tokens and asks the model to reconstruct them — forcing rich bidirectional representations.
- **NSP (optional):** an auxiliary task that teaches relationships between sentence pairs (useful for discourse-level tasks).

3 Masked Language Modeling (MLM)

3.1 Definition and purpose

Masked Language Modeling (MLM) randomly selects token positions in an input sequence, corrupts them (often by replacing with `[MASK]` or a random token), and trains the encoder to recover the original tokens. Predicting missing tokens requires integrating both left and right context.

3.2 Masking procedure (BERT's 15% rule)

1. Randomly choose 15% of token positions to form the mask set M.
2. For each chosen position $i \in M$:
   - with probability **0.80** replace the token with `[MASK]`;
   - with probability **0.10** replace the token with a random token from the vocabulary;
   - with probability **0.10** keep the original token unchanged.
3. Feed the corrupted sequence into the encoder and compute predictions only at indices $i \in M$.

3.3 Mathematical objective

Let the original token sequence be $x = (x_1, \ldots, x_N)$ and M the masked index set. After passing the corrupted input through the encoder we obtain contextual vectors h_i for each position. The MLM loss is:

$$L_{MLM} = -\frac{1}{|M|} \sum_{i \in M} \log P(x_i \mid h_i).$$

Only positions in M contribute to the loss, but *all* tokens participate in the attention computations and influence each other.

3.4 Why the 80/10/10 split?

- Always using `[MASK]` creates a train-test mismatch: at inference there are no `[MASK]` tokens. The 10% choices reduce reliance on the special token.
- Random replacements teach the model to detect contextually unlikely tokens and prefer semantically-fitting options.
- Leaving tokens unchanged regularizes the model by forcing it to predict even when the true token remains present.

3.5 Worked example (step-by-step)

Original sentence:
"The food was delicious today."
Suppose positions 2 ("food") and 4 ("delicious") are chosen as M. Apply the 80/10/10 rule:

- position 2 (food): replaced with `[MASK]` (80%)
- position 4 (delicious): replaced by a random token "planet" (10%)

Corrupted input:
"The [MASK] was planet today."
The encoder computes contextual vectors $h_1, \dots, h_5$. The MLM loss for these positions is:

$$L_{MLM} = -\frac{1}{2}\big(\log P(\text{food} \mid h_2) + \log P(\text{delicious} \mid h_4)\big).$$

If the model assigns high probability to the true tokens, the loss is small (good).

3.6 Intuition: what does the contextual vector h_i encode?

h_i is a compact vector representing the model's beliefs about the needed information at position i. It mixes signals about syntax (e.g., noun vs. verb), semantics (topic, named entities), and long-range relations (coreference, discourse). The vocabulary projection converts h_i into logits over tokens.

4 Bidirectional attention: removing the causal mask

4.1 Contrast with causal (autoregressive) models

- **Causal decoder (GPT-like):** attention mask enforces $j \le i$; position i cannot see future tokens. This is required for autoregressive generation.
- **BERT encoder:** no causal mask. The attention score matrix S with elements $S_{ij} = Q_i \cdot K_j / \sqrt{d_k}$ is fully visible; each token can attend to any other token.

4.2 Practical consequences

- The attention computation and memory scale as $O(N^2)$ for sequence length N.
- Because the entire sequence is available at once, KV caching (used for autoregressive inference) is unnecessary.
- Bidirectional attention produces stronger representations for understanding tasks but is computationally heavier for long contexts.

5 Next Sentence Prediction (NSP) — optional task

5.1 Motivation

NSP is an auxiliary task used in BERT's original pretraining: given segments (A,B) predict whether B is the true continuation of A. This helps the model learn discourse-level relations.

5.2 Input formatting

Inputs are formed as:

`[CLS]` A `[SEP]` B `[SEP]`

Segment embeddings indicate whether a token belongs to A (segment 0) or B (segment 1).

5.3 NSP head and loss

Use the `[CLS]` final vector h_{CLS}. Apply a linear classifier:

$$P(\text{IsNext} \mid h_{CLS}) = \text{softmax}(h_{CLS} W_{NSP}),$$

where $W_{NSP} \in \mathbb{R}^{d \times 2}$. The NSP loss is the cross-entropy over the two classes.

Note: Later models (e.g., RoBERTa) removed NSP and relied on more MLM data instead.

6 Architecture details

6.1 Input representation

Each token's input vector is the sum of:

- Token embedding (from embedding matrix E),
- Positional embedding (learned),
- Segment embedding (0 or 1).

The summed vector has dimension d and is fed to the transformer stack.

6.2 Transformer encoder block — components

1. Multi-head self-attention: project inputs to Q,K,V for each head, compute scaled dot-product attention, concatenate heads, and apply an output linear projection.
2. Residual connection + LayerNorm.
3. Position-wise feed-forward (two linear layers with GELU activation).
4. Residual connection + LayerNorm.

6.3 Common model sizes

- **BERT-Base:** $d = 768$, $L = 12$, $H = 12$, vocab ≈30k, context $N = 512$.
- **XLM-R (Base):** $d = 1024$, $L = 24$, $H = 16$, vocab ≈250k, context $N = 512$.

7 Training pipeline and practical tips

7.1 Optimization and scheduling

- Optimizer: AdamW (weight decay helps regularize).
- Learning rate: use linear warmup followed by linear decay.
- Mixed precision (AMP) recommended for speed and memory savings.

7.2 Data packing and batching

- Pack multiple short documents into a single context window to maximize GPU utilization.
- Insert explicit separator tokens between packed documents to avoid unwanted context mixing.

7.3 Masking and implementation notes

Implementation tips:

- Pre-sample masked positions before packing to guarantee correct overall masking frequency.
- Avoid using exact -INF for attention masking in mixed precision; use a large negative constant (e.g., -1e9).
- If training with very large vocabularies, consider tying token embeddings and output projection to reduce parameter count.

8 Worked numerical example (tiny model)

We compute attention and the MLM loss on a tiny toy example.
Vocabulary: "The", "cat", "sat", "on", "mat" (size 5).
Sequence: "The cat sat" (N=3). Suppose we mask position 2 ("cat").
Embeddings (toy):

$$e(\text{The}) = [1, 0, 0], \quad e(\text{cat}) = [0, 1, 0], \quad e(\text{sat}) = [0, 0, 1].$$

Assume single-head attention with $W_Q = W_K = W_V = I$ and $d_{head} = 3$. Then Q,K,V equal embeddings.
Scores for query at position 2 against keys 1..3 (dot products):

$$s_1 = Q_2 \cdot K_1 = 0, \quad s_2 = 1, \quad s_3 = 0.$$

Softmax over [0,1,0] gives weights approximately [0.2447, 0.5106, 0.2447].
Attention output (weighted sum of V vectors):

$$attn_2 = 0.2447e(\text{The}) + 0.5106e(\text{cat}) + 0.2447e(\text{sat}) \approx [0.2447, 0.5106, 0.2447].$$

Pass this through a tiny vocabulary head and softmax to get probabilities; the MLM loss for the masked position is $-\log P(\text{cat})$.

9 Exercises and short projects

1. Implement an MLM data generator (80/10/10) and validate the empirical frequencies.
2. Train a tiny transformer encoder on a toy corpus with MLM objective; evaluate embeddings by nearest-neighbor retrieval for masked token prediction.
3. Run ablations with and without NSP to observe effects on downstream sentence-pair tasks.

10 Summary checklist

- MLM: mask 15% tokens; model reconstructs originals using bidirectional context.
- NSP: optional; binary classification on sentence pairs using the `[CLS]` vector.
- BERT encoders use fully visible attention and produce contextual token vectors h_i.
- Implementation tips: avoid exact -INF, use warmup, mix domains, and pack documents carefully.

Want diagrams, runnable PyTorch notebooks, or a merged Sections 1–4 + post-training doc? Tell me which and I will add them directly into this LaTeX file.