Natural Language Inference

Iuliu Andrei Steau

June 2, 2025

SNLI

SNLI (Stanford Natural Language Inference) is a large-scale dataset for training and evaluating models on *natural language inference (NLI)*.

Dataset Split:

- **Training:** 550,152 pairs
- **Development:** 10,000 pairs
- **Test:** 10,000 pairs

Each example includes:

- A **premise** and a **hypothesis**
- A label:
  - `entailment`
  - `contradiction`
  - `neutral`
  - `-1` (no consensus / unclear)

Example:

- **Premise:** A man is playing a guitar.
- **Hypothesis:** A man is making music.
- **Label:** `entailment`

What is RoBERTa?

RoBERTa Overview

- **Robustly Optimized BERT Approach** - Facebook AI's enhanced transformer
- **Transformer Architecture:** Based on Encoder only architecture
- **Bidirectional Language Model:** Understands context from both directions
- **Pre-trained for Fine-tuning:** Ready for downstream NLP tasks

Why Choose RoBERTa?

- **Superior Performance:** Consistently outperforms BERT
- **Better Generalization:** Robust across diverse domains
- **Easy Integration:** Drop-in BERT replacement
- **Optimized Training:** More efficient learning process

RoBERTa Strengths in NLI

- **Top SNLI Results:** 92.3% - 93.1% accuracy (best 2 results)
- **Strong Cross-Domain Transfer:** SNLI & MNLI >90%
- **Pre-trained Models:** Ready-to-use NLI variants available

RoBERTa Architecture

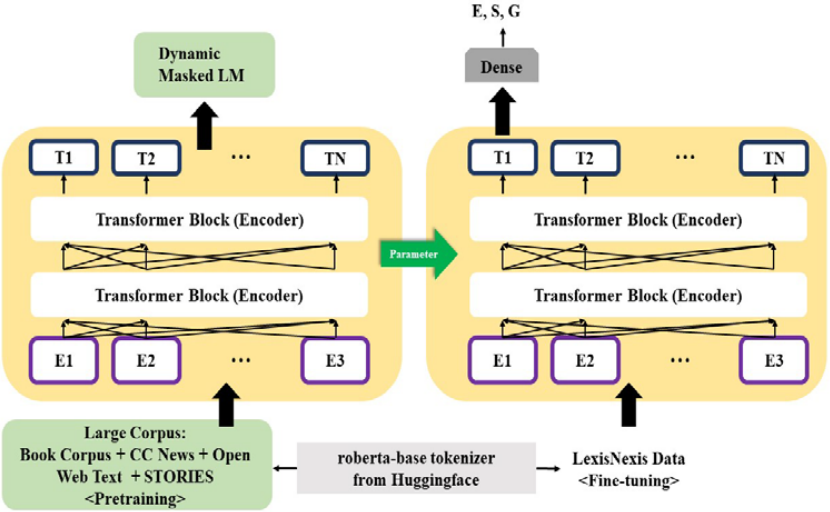


Figure: RoBERTa Architecture

RoBERTa vs BERT: Key Differences

Training Data Improvements

- **10x More Data:** 160GB training corpus vs BERT's 16GB
- **Diverse Sources:** CommonCrawl, OpenWebText, News articles
- **Longer Training:** More iterations and larger mini-batches

Training Procedure Changes

- **Dynamic Masking:** Different tokens masked per epoch vs static masking
- **Removed NSP:** No Next Sentence Prediction task (focus on MLM only)
- **Larger Batches:** 8K sequences vs BERT's smaller batches
- **Enhanced Vocabulary:** 50K BPE tokens vs BERT's 30K

Performance Improvements

- **Better Accuracy:** Higher scores on GLUE, SQuAD, RACE benchmarks
- **More Robust:** Better handling of linguistic variations
- **Faster Convergence:** More efficient training process

First Architecture

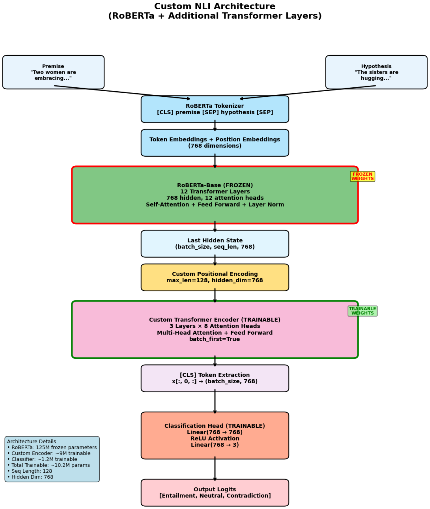
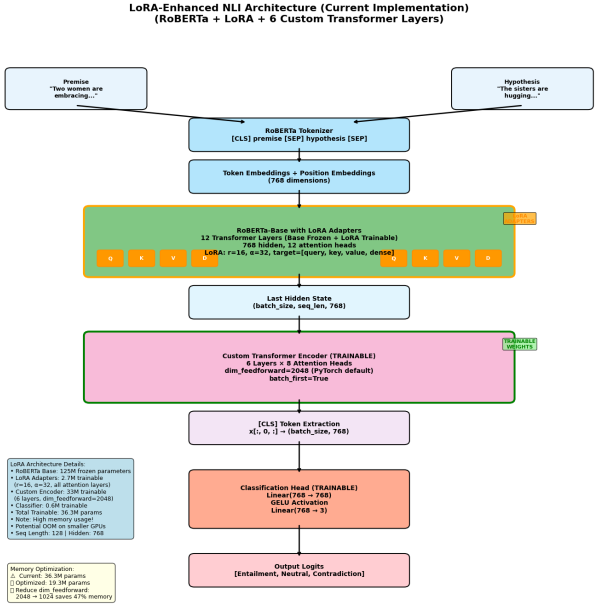


Figure: RoBERTa + 3 TransformerEncoder Architecture

First Architecture

- **Slow training:** around 45 min for an epoch !!!
- **Smaller number of parameters:** around 10.2 M
- **Smaller batches:** 32 samples/batch
- **Epochs:** Trained only over 5 epochs
- **Test accuracy** $\sim$ 82%

Second Architecture



Second Architecture

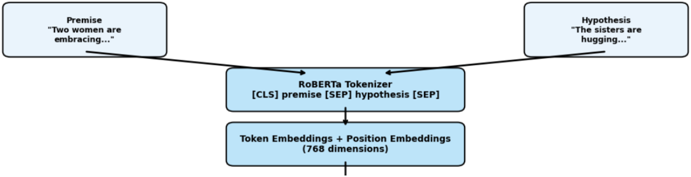


Figure: LoRA-Enhanced RoBERTa for NLI

Second Architecture

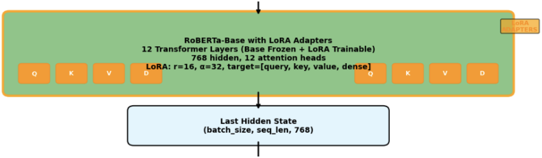


Figure: LoRA-Enhanced RoBERTa for NLI

Second Architecture

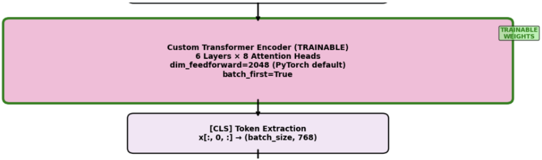


Figure: LoRA-Enhanced RoBERTa for NLI

Second Architecture

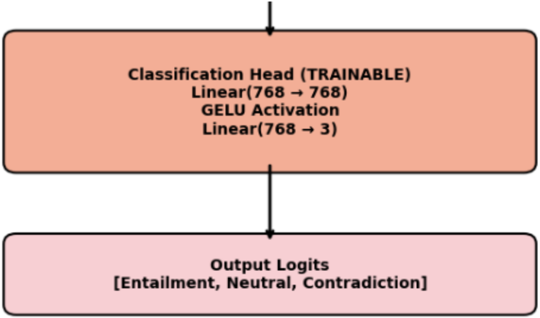


Figure: LoRA-Enhanced RoBERTa for NLI

Second Architecture

- **Fast training:** around 5$\sim$6 min for an epoch !!!
- **LoRA adaptation:** Train every W_q,W_k,W_v,W_d (around 2.7M trainable parameters)
- **Highly optimized:** Trained on A100 with multiple optimization - 15 times faster(details later)
- **High number of parameters:** $\sim$ 36.3 M
- **Robust activation function:** GELU allows negative values close to 0 to be propagated and is continuous at 0
- **Big batches:** 512 samples/batch
- **Epochs:** Trained on 15 epochs
- **Test accuracy** $\sim$ 90%

LoRA (Low-Rank Adaptation of Large Language Models)

How LoRa works

- **Parameter-Efficient Fine-tuning:** Adapts pre-trained models with minimal parameters using `output = input × (W +` ΔW`)`
- **Low-Rank Decomposition:** $\Delta W = BA$ where $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$
- **Mathematical Foundation:** Decomposes weight updates into low-rank matrices
- **Trainable Parameters:** Only A and B matrices, original weights W remain frozen

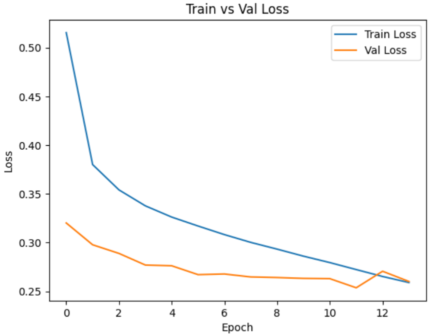
LoRA Configuration Details

- **Rank r=16:** Bottleneck dimension, much smaller than original (768×768)
- **Scaling Factor =32:** Controls adaptation strength: $\alpha / r \times \Delta W$
- **Target Matrices:** Query, Key, Value, Dense projections in attention layers
- **Dropout:** Applied to LoRA layers for regularization during training

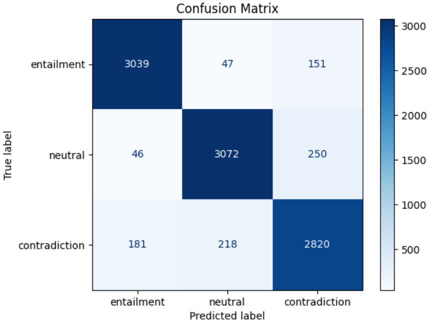
Key Advantages

- **Storage Efficient:** Small adapter files (few MB vs full model GB)
- **Task Modularity:** Switch between different task-specific adapters
- **Reduced Overfitting:** Fewer parameters prevent memorization
- **Hardware Friendly:** Enables fine-tuning on consumer GPUs

Results



Train vs Val Loss
Train Loss
Val Loss
Loss
Epoch



Confusion Matrix
True label
Predicted label
entailment
neutral
contradiction
3039
47
151
46
3072
250
181
218
2820

Optimizations

torch.compile() Optimization

- **TorchDynamo JIT:** Bytecode interception captures Python execution, extracts FX graphs for TorchInductor backend compilation
- **Operator Fusion & Scheduling:** Fuses ElementWise+MatMul+Activation chains, applies loop tiling and memory coalescing optimizations
- **Graph Specialization:** `compiled_model = torch.compile(model)` triggers lazy compilation with shape/dtype specialization
- **Autograd Integration:** Preserves gradient computation through FX graph transformations and backward pass optimization

Optimizations

Mode: "max-autotune-no-cudagraphs"

- **Triton Autotuning:** Exhaustive kernel parameter search (block_size, num_warps, num_stages) for optimal SM utilization
- **Shape Polymorphism:** Avoids CUDA graph static constraints, supports dynamic seq_len without recompilation overhead
- **Memory Allocator Bypass:** Uses PyTorch memory pool instead of CUDA graph capture, prevents memory fragmentation issues
- **Attention Kernel Optimization:** Flash Attention v2 integration with optimal tile sizes for transformer workloads (1.2x-1.8x speedup)
- **LoRA Pattern Recognition:** Detects low-rank decomposition patterns, fuses frozen_weight + (B @ A) operations in single kernel

Optimizations

Mixed Precision Training

- **BFloat16 Autocast:** `torch.amp.autocast(device_type='cuda', dtype=torch.bfloat16)` for modern GPU training
- **TensorFloat-32 (TF32):** `torch.backends.cuda.matmul.allow_tf32 = True` enables accelerated matrix multiplications on Ampere GPUs
- **cuDNN TF32:** `torch.backends.cudnn.allow_tf32 = True` accelerates convolution and normalization operations
- **Flash Attention:** `torch.backends.cuda.enable_flash_sdp(True)` enables memory-efficient attention computation
- **Gradient Scaler:** `scaler = GradScaler()` prevents gradient underflow in mixed precision training
- **Memory Efficiency:** 50% memory reduction with 1.5-2x speedup on transformer workloads

Optimizations

Hardware Acceleration Details

- **TF32 Benefits:** 19-bit precision (vs FP32's 23-bit) with 8x throughput improvement on A100/H100
- **Flash Attention Optimization:** O(N) memory complexity vs O(N²) for standard attention, ideal for long sequences
- **Automatic Precision:** BF16 forward pass, FP32 gradient accumulation with dynamic loss scaling
- **LoRA Acceleration:** TF32 matrix multiplications particularly benefit low-rank decomposition operations

Optimizations

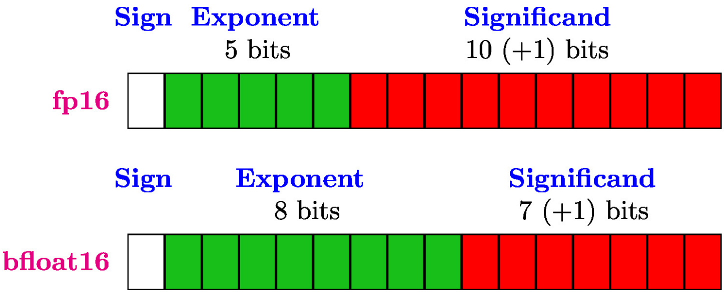


Figure: BFloat16 Mixed Precision Architecture

Optimizations

Hardware Comparison: A100 vs L4

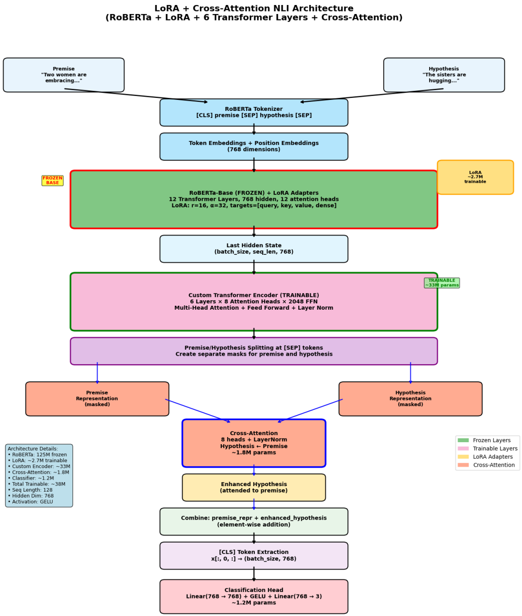
- **Compute Capability:** A100: 108 SMs, 40GB/80GB HBM2e vs L4: 58 SMs, 24GB GDDR6
- **Memory Bandwidth:** A100: 1935 GB/s HBM2e vs L4: 300 GB/s GDDR6 (6.5x difference)
- **Tensor Core Performance:** A100: 312 TFLOPS BF16 vs L4: 121 TFLOPS BF16 (2.6x difference)
- **TF32 Acceleration:** A100: 156 TFLOPS vs L4: 60 TFLOPS for matrix operations
- **Power Consumption:** A100: 400W TDP vs L4: 72W TDP (5.5x more efficient per watt)

Optimizations

Training Performance Implications

- **Batch Size:** A100 supports larger batches (40-80GB) vs L4 limited to smaller batches (24GB)
- **LoRA Training:** L4 sufficient for LoRA (2.7M parameters) but slower than A100 for full models
- **Memory Efficiency:** L4 requires aggressive optimization (BF16, gradient checkpointing, cache cleanup)
- **Flash Attention:** More critical on L4 due to memory constraints, provides larger relative speedup
- **Cost Efficiency:** L4 offers better price/performance for parameter-efficient fine-tuning workflows

Attempt


LoRA + Cross-Attention NLI Architecture
(RoBERTa + LoRA + 6 Transformer Layers + Cross-Attention)
Premise
"Two women are embracing..."
Hypothesis
"The sisters are hugging..."
RoBERTa Tokenizer
[CLS] premise [SEP] hypothesis [SEP]
Token Embeddings + Position Embeddings
(768 dimensions)
FROZEN BASE
RoBERTa-Base (FROZEN) + LoRA Adapters
12 Transformer Layers, 768 hidden, 12 attention heads
LoRA: r=16, α=32, targets=[query, key, value, dense]
LoRA
~2.7M
trainable
Last Hidden State
(batch_size, seq_len, 768)
TRAINABLE
~33M params
Custom Transformer Encoder (TRAINABLE)
6 Layers × 8 Attention Heads × 2048 FFN
Multi-Head Attention + Feed Forward + Layer Norm
Premise/Hypothesis Splitting at [SEP] tokens
Create separate masks for premise and hypothesis
Premise
Representation
(masked)
Hypothesis
Representation
(masked)
Cross-Attention
8 heads + LayerNorm
Hypothesis ← Premise
~1.8M params
Architecture Details:
• RoBERTa: 125M frozen
• LoRA: ~2.7M trainable
• Custom Encoder: ~33M
• Cross-Attention: ~1.8M
• Classifier: ~1.2M
• Total Trainable: ~38M
• Seq Length: 128
• Hidden Dim: 768
• Activation: GELU
Frozen Layers
Trainable Layers
LoRA Adapters
Cross-Attention
Enhanced Hypothesis
(attended to premise)
Combine: premise_repr + enhanced_hypothesis
(element-wise addition)
[CLS] Token Extraction
x[:, 0, :] → (batch_size, 768)
Classification Head
Linear(768 → 768) + GELU + Linear(768 → 3)
~1.2M params

Attempt

Key Architectural Changes

- **Cross-Attention Module:** Added dedicated 8-head cross-attention for premise-hypothesis interaction (1.8M params)
- **Explicit Separation:** Premise and hypothesis representations split after joint encoding instead of single [CLS] token
- **Enhanced Fusion:** Element-wise combination of premise + cross-attended hypothesis vs direct [CLS] classification
- **Interpretable Reasoning:** Attention weights reveal premise-hypothesis alignment patterns for better NLI understanding

Unsolved issue

- **Stagnant Loss:** Kept around 1.1 ($\log(3) \sim 1.1$ from Cross-Entropy)

Errors

torch.compile() Related Errors

- **Graph Recording Error:** "input tensor deallocate during graph recording that did not occur during replay"
- **Solution:** Use `mode="max-autotune-no-cudagraphs"` to avoid CUDA graph memory issues
- **TorchDynamo Compilation:** Extensive bytecode transformation logs during first compilation pass
- **Mitigation:** Expected behavior - compilation happens once, then caches optimized version

Errors

Tensor Dimension Mismatches

- **Size Error:** "Expected size 16384 but got size 16" in tensor concatenation operations
- **Root Cause:** Batch size or sequence length inconsistencies between model components
- **Solution:** Verify tensor shapes at each layer: RoBERTa output → Custom Transformer → Cross-Attention
- **Debug Strategy:** Add shape logging: `print(f"Tensor shape: {tensor.shape}")` at critical points

Errors

CUDA & Memory Errors

- **Device-Side Assert:** CUDA kernel errors with device-side assertion triggers
- **Solution:** Compile with `TORCH_USE_CUDA_DSA=1` for better debugging information
- **Model Initialization:** RobertaModel weights not properly initialized from checkpoint
- **Fix:** Ensure proper model loading sequence: base model $\rightarrow$ LoRA adapters $\rightarrow$ custom layers

Errors

Prevention Best Practices

- **Gradual Compilation:** Test without torch.compile() first, then add optimization
- **Memory Management:** Regular `torch.cuda.empty_cache()` calls during training
- **Error Handling:** Wrap training loops in try-except for graceful error recovery

References

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov.
RoBERTa: A Robustly Optimized BERT Pretraining Approach.
arXiv preprint arXiv:1907.11692, 2019.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova.
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
arXiv preprint arXiv:1810.04805, 2018.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin.
Attention is All You Need.
arXiv preprint arXiv:1706.03762, 2017.

S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning.
A large annotated corpus for learning natural language inference.
In *Proceedings of EMNLP*, pages 632–642, 2015.

A. Williams, N. Nangia, and S. R. Bowman.
A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference.
In *Proceedings of NAACL-HLT*, pages 1112–1122, 2018.