AT82.05 Artificial Intelligence: Natural Language Understanding (NLU)

A4: Do You AGREE

Name: Arya Shah

StudentID: st125462

In this assignment, I will explore training a pre-trained model like BERT from scratch, focusing on leveraging text embeddings to capture semantic similarity. Additionally, we will explore how to adapt the loss function for tasks like Natural Language Inference (NLI) to enhance the model's ability to understand semantic relationships between texts

You can find the GitHub Repository for the assignment here:

- https://github.com/aryashah2k/NLP-NLU (Complete Web App)
- https://github.com/aryashah2k/NLP-NLU/tree/main/notebooks (Assignment Notebooks)
- https://github.com/aryashah2k/NLP-NLU/tree/main/reports (Assignment Reports)

Task 1: Training BERT from Scratch ✅

Based on Masked Language Model/BERT-update.ipynb, modify as follows: (2 points)

1. Implement Bidirectional Encoder Representations from Transformers (BERT) from scratch, following the concepts learned in class. ✅
2. Train the model on a suitable dataset. Ensure to source this dataset from reputable public databases or repositories. It is imperative to give proper credit to the dataset source in your documentation. ✅
3. Save the trained model weights for later use in Task 2. ✅

In [ ]:
```
### Dataset Source Attribution and Credits

"""
Dataset Documentation

1. BookCorpus Dataset
-------------------
Description:
    A large collection of free novel books written by unpublished authors.
    Contains approximately 74M sentences and 1B words from 11,038 books

Usage:
    from datasets import load_dataset
```

```
    # Load full dataset
    dataset = load_dataset('bookcorpus')

    # Load specific splits
    train_test = load_dataset('bookcorpus', split='train+test')

    # Load percentage of data
    partial_data = load_dataset('bookcorpus', split='train[:10%]')[1]

2. SNLI (Stanford Natural Language Inference) Dataset
-------------------------------------------------
Description:
    A collection of 570k human-written English sentence pairs labeled for
    balanced classification with entailment, contradiction, and neutral labels.

Structure:
    - Total Instances: 570,152
    - Splits:
        - Train: 550,152
        - Validation: 10,000
        - Test: 10,000

Data Fields:
    - premise: str  # Base statement for comparison
    - hypothesis: str  # Statement to be evaluated against premise
    - label: int  # 0: entailment, 1: neutral, 2: contradiction, -1: no consensu

Average Token Counts:
    - Premise: 14.1 tokens
    - Hypothesis: 8.3 tokens

Usage:
    from datasets import load_dataset

    dataset = load_dataset('stanfordnlp/snli')

    # Filter invalid labels
    valid_data = dataset.filter(lambda x: x['label'] != -1)

Note:
    Each premise appears in only one split, though it may be used in
    multiple examples with different hypotheses
"""
```

In [6]:

```
import math
import numpy as np
import re
import logging
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from datasets import load_dataset
from tqdm.auto import tqdm
import os
import json
from datetime import datetime
from transformers import get_cosine_schedule_with_warmup

# Configure logging
```

```python
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('bert_training.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

def get_free_gpu():
    """Get the GPU with the most available memory."""
    if not torch.cuda.is_available():
        return torch.device('cpu')

    # Get the number of GPUs
    n_gpus = torch.cuda.device_count()
    if n_gpus == 0:
        return torch.device('cpu')

    # Find GPU with most free memory
    max_free_memory = 0
    selected_gpu = 0

    for gpu_id in range(n_gpus):
        try:
            # Get free memory for this GPU
            free_memory = torch.cuda.get_device_properties(gpu_id).total_memory
            if free_memory > max_free_memory:
                max_free_memory = free_memory
                selected_gpu = gpu_id
        except:
            continue

    device = torch.device(f'cuda:{selected_gpu}')
    logger.info(f"Selected GPU {selected_gpu} with {max_free_memory/1024/1024:.2
    return device

# Set device and seeds for reproducibility
SEED = 1234
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
device = get_free_gpu()
logger.info(f"Using device: {device}")

# Model configuration
class BertConfig:
    def __init__(self):
        # Model architecture
        self.vocab_size = None  # Will be set after data loading
        self.hidden_size = 256
        self.num_hidden_layers = 6
        self.num_attention_heads = 8
        self.intermediate_size = 1024

        # Dropout and normalization
        self.hidden_dropout_prob = 0.1
        self.attention_probs_dropout_prob = 0.1
        self.layer_norm_eps = 1e-12
```

```python
        # Sequence parameters
        self.max_position_embeddings = 128
        self.max_len = 128
        self.type_vocab_size = 2
        self.pad_token_id = 0

        # Special tokens
        self.mask_token_id = 3
        self.cls_token_id = 1
        self.sep_token_id = 2
        self.pad_token_id = 0

        # Training parameters
        self.learning_rate = 1e-4
        self.batch_size = 64
        self.gradient_accumulation_steps = 4
        self.weight_decay = 0.01
        self.adam_epsilon = 1e-8
        self.warmup_ratio = 0.1

class BertLayerNorm(nn.Module):
    def __init__(self, hidden_size, eps=1e-12):
        super().__init__()
        self.weight = nn.Parameter(torch.ones(hidden_size))
        self.bias = nn.Parameter(torch.zeros(hidden_size))
        self.variance_epsilon = eps

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        variance = (x - mean).pow(2).mean(-1, keepdim=True)
        x = (x - mean) / torch.sqrt(variance + self.variance_epsilon)
        return self.weight * x + self.bias

class BertEmbeddings(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.word_embeddings = nn.Embedding(config.vocab_size, config.hidden_siz
        self.position_embeddings = nn.Embedding(config.max_position_embeddings, 
        self.token_type_embeddings = nn.Embedding(config.type_vocab_size, config
        self.LayerNorm = BertLayerNorm(config.hidden_size, eps=config.layer_norm
        self.dropout = nn.Dropout(config.hidden_dropout_prob)

    def forward(self, input_ids, token_type_ids=None, position_ids=None):
        seq_length = input_ids.size(1)
        if position_ids is None:
            position_ids = torch.arange(seq_length, dtype=torch.long, device=inp
            position_ids = position_ids.unsqueeze(0).expand_as(input_ids)
        if token_type_ids is None:
            token_type_ids = torch.zeros_like(input_ids)

        words_embeddings = self.word_embeddings(input_ids)
        position_embeddings = self.position_embeddings(position_ids)
        token_type_embeddings = self.token_type_embeddings(token_type_ids)

        embeddings = words_embeddings + position_embeddings + token_type_embeddi
        embeddings = self.LayerNorm(embeddings)
        embeddings = self.dropout(embeddings)
        return embeddings

class BertSelfAttention(nn.Module):
```

```python
    def __init__(self, config):
        super().__init__()
        self.num_attention_heads = config.num_attention_heads
        self.attention_head_size = config.hidden_size // config.num_attention_he
        self.all_head_size = self.num_attention_heads * self.attention_head_size

        # Initialize with smaller values for stability
        self.query = nn.Linear(config.hidden_size, self.all_head_size)
        self.key = nn.Linear(config.hidden_size, self.all_head_size)
        self.value = nn.Linear(config.hidden_size, self.all_head_size)

        self.dropout = nn.Dropout(config.attention_probs_dropout_prob)
        self.layer_norm = BertLayerNorm(config.hidden_size, eps=config.layer_nor

        # Initialize weights
        self._init_weights()

    def _init_weights(self):
        for module in [self.query, self.key, self.value]:
            module.weight.data.normal_(mean=0.0, std=0.02)
            if module.bias is not None:
                module.bias.data.zero_()

    def transpose_for_scores(self, x):
        new_x_shape = x.size()[:-1] + (self.num_attention_heads, self.attention_
        x = x.view(*new_x_shape)
        return x.permute(0, 2, 1, 3)

    def forward(self, hidden_states, attention_mask=None):
        query_layer = self.transpose_for_scores(self.query(hidden_states))
        key_layer = self.transpose_for_scores(self.key(hidden_states))
        value_layer = self.transpose_for_scores(self.value(hidden_states))

        attention_scores = torch.matmul(query_layer, key_layer.transpose(-1, -2)
        attention_scores = attention_scores / math.sqrt(self.attention_head_size

        if attention_mask is not None:
            attention_scores = attention_scores + attention_mask

        attention_probs = nn.Softmax(dim=-1)(attention_scores)
        attention_probs = self.dropout(attention_probs)

        context_layer = torch.matmul(attention_probs, value_layer)
        context_layer = context_layer.permute(0, 2, 1, 3).contiguous()
        new_context_layer_shape = context_layer.size()[:-2] + (self.all_head_siz
        context_layer = context_layer.view(*new_context_layer_shape)

        return context_layer

class BertLayer(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.attention = BertSelfAttention(config)
        self.intermediate = nn.Linear(config.hidden_size, config.intermediate_si
        self.output = nn.Linear(config.intermediate_size, config.hidden_size)
        self.LayerNorm1 = BertLayerNorm(config.hidden_size, eps=config.layer_nor
        self.LayerNorm2 = BertLayerNorm(config.hidden_size, eps=config.layer_nor
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.activation = F.gelu
```

```python
    def forward(self, hidden_states, attention_mask=None):
        attention_output = self.attention(hidden_states, attention_mask)
        attention_output = self.dropout(attention_output)
        attention_output = self.LayerNorm1(attention_output + hidden_states)

        intermediate_output = self.intermediate(attention_output)
        intermediate_output = self.activation(intermediate_output)

        layer_output = self.output(intermediate_output)
        layer_output = self.dropout(layer_output)
        layer_output = self.LayerNorm2(layer_output + attention_output)

        return layer_output

class BertModel(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.config = config
        self.embeddings = BertEmbeddings(config)
        self.encoder = nn.ModuleList([BertLayer(config) for _ in range(config.nu
        self.pooler = nn.Linear(config.hidden_size, config.hidden_size)
        self.pooler_activation = nn.Tanh()

        # MLM head
        self.mlm_head = nn.Linear(config.hidden_size, config.vocab_size)

        # Enable gradient checkpointing for memory efficiency
        self.gradient_checkpointing = False

    def forward(self, input_ids, token_type_ids=None, attention_mask=None, maske
        if attention_mask is None:
            attention_mask = torch.ones_like(input_ids)

        extended_attention_mask = attention_mask.unsqueeze(1).unsqueeze(2)
        extended_attention_mask = extended_attention_mask.to(dtype=next(self.par
        extended_attention_mask = (1.0 - extended_attention_mask) * -10000.0

        embedding_output = self.embeddings(input_ids, token_type_ids)

        hidden_states = embedding_output

        for layer in self.encoder:
            if self.gradient_checkpointing and self.training:
                def create_custom_forward(module):
                    def custom_forward(*inputs):
                        return module(*inputs)
                    return custom_forward

                hidden_states = torch.utils.checkpoint.checkpoint(
                    create_custom_forward(layer),
                    hidden_states,
                    extended_attention_mask,
                )
            else:
                hidden_states = layer(hidden_states, extended_attention_mask)

        # MLM loss calculation
        if masked_lm_labels is not None:
            prediction_scores = self.mlm_head(hidden_states)
            loss_fct = nn.CrossEntropyLoss(ignore_index=-1)
```

```
            masked_lm_loss = loss_fct(prediction_scores.view(-1, self.config.voc
                                      masked_lm_labels.view(-1))
            return masked_lm_loss

        return hidden_states

    def enable_gradient_checkpointing(self):
        self.gradient_checkpointing = True

def pad_sequence(tokens, max_len, pad_token):
    """Pad or truncate a sequence to max_len."""
    if len(tokens) > max_len:
        return tokens[:max_len]
    return tokens + [pad_token] * (max_len - len(tokens))

def prepare_batch(texts, word2idx, config):
    """Prepare a batch of texts for BERT training."""
    # Convert texts to token ids and pad
    batch_input_ids = []
    for text in texts:
        tokens = text.split()
        token_ids = [word2idx.get(word, word2idx['[UNK]']) for word in tokens]
        # Add [CLS] at start and [SEP] at end
        token_ids = [word2idx['[CLS]']] + token_ids + [word2idx['[SEP]']]
        # Pad sequence
        padded_ids = pad_sequence(token_ids, config.max_len, word2idx['[PAD]'])
        batch_input_ids.append(padded_ids)

    # Convert to tensor
    input_ids = torch.tensor(batch_input_ids).to(device)
    attention_mask = (input_ids != word2idx['[PAD]']).float()

    return input_ids, attention_mask

def load_and_preprocess_data():
    logger.info("Loading BookCorpus dataset...")
    # Load only 100k samples as specified
    dataset = load_dataset('bookcorpus', split='train[:150000]')

    logger.info("Preprocessing text data...")
    texts = dataset['text']
    texts = [text.lower() for text in texts]
    texts = [re.sub(r'[^\w\s]', '', text) for text in texts]

    # Create vocabulary
    logger.info("Creating vocabulary...")
    word_set = set()
    for text in texts:
        words = text.split()
        word_set.update(words)

    # Add special tokens
    vocab = ['[PAD]', '[CLS]', '[SEP]', '[MASK]', '[UNK]'] + list(word_set)
    word2idx = {word: idx for idx, word in enumerate(vocab)}

    return texts, word2idx, vocab

def save_model_and_config(model, config, epoch, loss, save_dir='model_checkpoint
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)
```

```
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    model_path = os.path.join(save_dir, f'bert_epoch_{epoch}_{timestamp}.pt')
    config_path = os.path.join(save_dir, f'config_{timestamp}.json')

    # Save model
    torch.save({
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'loss': loss,
    }, model_path)

    # Save config
    config_dict = {k: v for k, v in vars(config).items() if not k.startswith('__
    with open(config_path, 'w') as f:
        json.dump(config_dict, f, indent=4)

    logger.info(f"Model saved to {model_path}")
    logger.info(f"Config saved to {config_path}")

def main():
    logger.info("Starting BERT training from scratch")

    # Enable mixed precision training with proper initialization
    scaler = torch.amp.GradScaler(enabled=True)

    # Load and preprocess data
    texts, word2idx, vocab = load_and_preprocess_data()

    # Initialize config
    config = BertConfig()
    config.vocab_size = len(vocab)
    logger.info(f"Vocabulary size: {config.vocab_size}")

    # Initialize model with gradient checkpointing
    model = BertModel(config).to(device)
    model.enable_gradient_checkpointing()
    logger.info("Model initialized with gradient checkpointing")

    # Initialize optimizer with weight decay and proper learning rate
    no_decay = ['bias', 'LayerNorm.weight']
    optimizer_grouped_parameters = [
        {
            'params': [p for n, p in model.named_parameters() if not any(nd in n
            'weight_decay': 0.01
        },
        {
            'params': [p for n, p in model.named_parameters() if any(nd in n for
            'weight_decay': 0.0
        }
    ]
    optimizer = optim.AdamW(optimizer_grouped_parameters, lr=config.learning_rat

    # Add learning rate scheduler with warmup
    num_training_steps = len(texts) // (config.batch_size * config.gradient_accu
    num_warmup_steps = num_training_steps // 10
    scheduler = get_cosine_schedule_with_warmup(
        optimizer,
        num_warmup_steps=num_warmup_steps,
        num_training_steps=num_training_steps,
```

```
        num_cycles=0.5,
    )

    # Training loop
    logger.info("Starting training...")
    model.train()

    try:
        for epoch in range(15):
            total_loss = 0
            valid_loss_count = 0
            optimizer.zero_grad()  # Reset gradients at start of epoch

            progress_bar = tqdm(range(0, len(texts), config.batch_size),
                              desc=f"Epoch {epoch+1}")

            for step, batch_start in enumerate(progress_bar):
                batch_texts = texts[batch_start:batch_start + config.batch_size]

                # Prepare batch data with padding
                input_ids, attention_mask = prepare_batch(batch_texts, word2idx,

                # Create masked tokens
                masked_labels = input_ids.clone()
                special_tokens = {word2idx['[PAD]'], word2idx['[CLS]'], word2idx
                mask_candidates = torch.ones_like(input_ids, device=device).bool
                for special_token in special_tokens:
                    mask_candidates &= (input_ids != special_token)

                # Apply masking with 15% probability to valid tokens
                mask_prob = torch.full(input_ids.shape, 0.15, device=device)
                mask = (torch.bernoulli(mask_prob).bool() & mask_candidates)

                masked_labels[~mask] = -1  # Only compute loss on masked tokens
                input_ids[mask] = word2idx['[MASK]']

                try:
                    # Mixed precision forward pass
                    with torch.amp.autocast(device_type='cuda', dtype=torch.floa
                        loss = model(
                            input_ids,
                            attention_mask=attention_mask,
                            masked_lm_labels=masked_labels
                        )

                        # Scale loss for gradient accumulation
                        loss = loss / config.gradient_accumulation_steps

                        # Check if loss is valid
                        if not torch.isfinite(loss):
                            logger.warning(f"Non-finite loss detected: {loss.ite
                            continue

                    # Backward pass with gradient scaling
                    scaler.scale(loss).backward()

                    # Gradient accumulation
                    if (step + 1) % config.gradient_accumulation_steps == 0:
                        # Clip gradients
                        scaler.unscale_(optimizer)
```

```
                torch.nn.utils.clip_grad_norm_(model.parameters(), max_n

                # Optimizer step
                scaler.step(optimizer)
                scaler.update()
                scheduler.step()
                optimizer.zero_grad()

            # Update loss statistics
            loss_value = loss.item() * config.gradient_accumulation_step
            if np.isfinite(loss_value):
                total_loss += loss_value
                valid_loss_count += 1

            progress_bar.set_postfix({
                'loss': loss_value,
                'lr': scheduler.get_last_lr()[0]
            })

        except RuntimeError as e:
            if "out of memory" in str(e):
                logger.warning(f"Out of memory in batch. Skipping batch
                if hasattr(torch.cuda, 'empty_cache'):
                    torch.cuda.empty_cache()
                optimizer.zero_grad()
                continue
            raise e

        # Clear cache periodically
        if step % 100 == 0 and hasattr(torch.cuda, 'empty_cache'):
            torch.cuda.empty_cache()

    # Calculate average loss only from valid losses
    avg_loss = total_loss / valid_loss_count if valid_loss_count > 0 els
    logger.info(f"Epoch {epoch+1} completed. Average loss: {avg_loss:.4f

    # Save model checkpoint
    save_model_and_config(model, config, epoch+1, avg_loss)

  logger.info("Training completed!")

 except RuntimeError as e:
  if "out of memory" in str(e):
      logger.error(f"GPU out of memory error: {e}")
      logger.info("Try reducing batch_size or model size further if this e
      if hasattr(torch.cuda, 'empty_cache'):
          torch.cuda.empty_cache()
  raise e

if __name__ == "__main__":
    main()
```

```
2025-02-17 16:42:15,492 - INFO - Selected GPU 1 with 11004.50MB free memory
2025-02-17 16:42:15,494 - INFO - Using device: cuda:1
2025-02-17 16:42:15,502 - INFO - Starting BERT training from scratch
2025-02-17 16:42:15,503 - INFO - Loading BookCorpus dataset...
2025-02-17 16:42:19,577 - INFO - Preprocessing text data...
2025-02-17 16:42:20,363 - INFO - Creating vocabulary...
2025-02-17 16:42:20,579 - INFO - Vocabulary size: 27092
2025-02-17 16:42:20,990 - INFO - Model initialized with gradient checkpointing
2025-02-17 16:42:20,992 - INFO - Starting training...
Epoch 1:   0%|          | 0/2344 [00:00<?, ?it/s]
/home/jupyter-st125462/.local/lib/python3.12/site-packages/torch/_dynamo/eval_fra
me.py:632: UserWarning: torch.utils.checkpoint: the use_reentrant parameter shoul
d be passed explicitly. In version 2.5 we will raise an exception if use_reentran
t is not passed. use_reentrant=False is recommended, but if you need to preserve
the current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
2025-02-17 16:45:13,474 - INFO - Epoch 1 completed. Average loss: 7.9539 (from 23
44 valid batches)
2025-02-17 16:45:13,590 - INFO - Model saved to model_checkpoints/bert_epoch_1_20
250217_164513.pt
2025-02-17 16:45:13,591 - INFO - Config saved to model_checkpoints/config_2025021
7_164513.json
Epoch 2:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 16:48:06,279 - INFO - Epoch 2 completed. Average loss: 6.3747 (from 23
44 valid batches)
2025-02-17 16:48:06,383 - INFO - Model saved to model_checkpoints/bert_epoch_2_20
250217_164806.pt
2025-02-17 16:48:06,383 - INFO - Config saved to model_checkpoints/config_2025021
7_164806.json
Epoch 3:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 16:50:58,879 - INFO - Epoch 3 completed. Average loss: 6.1192 (from 23
44 valid batches)
2025-02-17 16:50:58,974 - INFO - Model saved to model_checkpoints/bert_epoch_3_20
250217_165058.pt
2025-02-17 16:50:58,975 - INFO - Config saved to model_checkpoints/config_2025021
7_165058.json
Epoch 4:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 16:53:50,245 - INFO - Epoch 4 completed. Average loss: 5.7795 (from 23
44 valid batches)
2025-02-17 16:53:50,346 - INFO - Model saved to model_checkpoints/bert_epoch_4_20
250217_165350.pt
2025-02-17 16:53:50,347 - INFO - Config saved to model_checkpoints/config_2025021
7_165350.json
Epoch 5:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 16:56:41,997 - INFO - Epoch 5 completed. Average loss: 5.5129 (from 23
44 valid batches)
2025-02-17 16:56:42,096 - INFO - Model saved to model_checkpoints/bert_epoch_5_20
250217_165641.pt
2025-02-17 16:56:42,097 - INFO - Config saved to model_checkpoints/config_2025021
7_165641.json
Epoch 6:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 16:59:33,606 - INFO - Epoch 6 completed. Average loss: 5.3713 (from 23
44 valid batches)
2025-02-17 16:59:33,713 - INFO - Model saved to model_checkpoints/bert_epoch_6_20
250217_165933.pt
2025-02-17 16:59:33,713 - INFO - Config saved to model_checkpoints/config_2025021
7_165933.json
```

```
Epoch 7:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 17:02:25,037 - INFO - Epoch 7 completed. Average loss: 5.2726 (from 2344 valid batches)
2025-02-17 17:02:25,143 - INFO - Model saved to model_checkpoints/bert_epoch_7_20250217_170225.pt
2025-02-17 17:02:25,144 - INFO - Config saved to model_checkpoints/config_20250217_170225.json
Epoch 8:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 17:05:16,252 - INFO - Epoch 8 completed. Average loss: 5.2281 (from 2344 valid batches)
2025-02-17 17:05:16,354 - INFO - Model saved to model_checkpoints/bert_epoch_8_20250217_170516.pt
2025-02-17 17:05:16,355 - INFO - Config saved to model_checkpoints/config_20250217_170516.json
Epoch 9:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 17:08:07,241 - INFO - Epoch 9 completed. Average loss: 5.2048 (from 2344 valid batches)
2025-02-17 17:08:07,347 - INFO - Model saved to model_checkpoints/bert_epoch_9_20250217_170807.pt
2025-02-17 17:08:07,347 - INFO - Config saved to model_checkpoints/config_20250217_170807.json
Epoch 10:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 17:10:58,332 - INFO - Epoch 10 completed. Average loss: 5.1990 (from 2344 valid batches)
2025-02-17 17:10:58,437 - INFO - Model saved to model_checkpoints/bert_epoch_10_20250217_171058.pt
2025-02-17 17:10:58,438 - INFO - Config saved to model_checkpoints/config_20250217_171058.json
Epoch 11:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 17:13:49,673 - INFO - Epoch 11 completed. Average loss: 5.2018 (from 2344 valid batches)
2025-02-17 17:13:49,778 - INFO - Model saved to model_checkpoints/bert_epoch_11_20250217_171349.pt
2025-02-17 17:13:49,779 - INFO - Config saved to model_checkpoints/config_20250217_171349.json
Epoch 12:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 17:16:40,828 - INFO - Epoch 12 completed. Average loss: 5.1833 (from 2344 valid batches)
2025-02-17 17:16:40,933 - INFO - Model saved to model_checkpoints/bert_epoch_12_20250217_171640.pt
2025-02-17 17:16:40,934 - INFO - Config saved to model_checkpoints/config_20250217_171640.json
Epoch 13:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 17:19:32,317 - INFO - Epoch 13 completed. Average loss: 5.1567 (from 2344 valid batches)
2025-02-17 17:19:32,422 - INFO - Model saved to model_checkpoints/bert_epoch_13_20250217_171932.pt
2025-02-17 17:19:32,423 - INFO - Config saved to model_checkpoints/config_20250217_171932.json
Epoch 14:   0%|          | 0/2344 [00:00<?, ?it/s]
2025-02-17 17:22:23,889 - INFO - Epoch 14 completed. Average loss: 5.1155 (from 2344 valid batches)
2025-02-17 17:22:23,993 - INFO - Model saved to model_checkpoints/bert_epoch_14_20250217_172223.pt
2025-02-17 17:22:23,994 - INFO - Config saved to model_checkpoints/config_20250217_172223.json
Epoch 15:   0%|          | 0/2344 [00:00<?, ?it/s]
```

```
2025-02-17 17:25:15,268 - INFO - Epoch 15 completed. Average loss: 5.0737 (from 2
344 valid batches)
2025-02-17 17:25:15,367 - INFO - Model saved to model_checkpoints/bert_epoch_15_2
0250217_172515.pt
2025-02-17 17:25:15,368 - INFO - Config saved to model_checkpoints/config_2025021
7_172515.json
2025-02-17 17:25:15,369 - INFO - Training completed!
```

Findings

[Training Logs can be found in the logs directory in the a4_do_you_agree/bert_training.log]

Hardware and Initialization

- Training used GPU 1 with approximately 11GB of free memory
- Device: CUDA-enabled GPU (cuda:1)

Dataset and Model Setup

- Successfully loaded BookCorpus dataset
- Vocabulary size: 27,092 tokens
- Model initialized with gradient checkpointing enabled

Training Progress

- Total epochs completed: 15
- Valid batches per epoch: 2,344
- Training duration: Approximately 43 minutes (16:42 to 17:25)

Loss Progression

Key loss values across epochs:

- Epoch 1: 7.9539
- Epoch 5: 5.5129
- Epoch 10: 5.1990
- Epoch 15: 5.0737 (final)

Model Performance Analysis

- Strong initial improvement: Loss dropped significantly from 7.95 to 6.37 between epochs 1-2
- Steady convergence: Loss continued to decrease gradually
- Final improvement: ~36% reduction in loss from start (7.95) to finish (5.07)

Technical Notes

- Model checkpoints and configurations were saved after each epoch
- Some widget display errors were logged but didn't affect training
- Warning about torch.utils.checkpoint parameter usage was recorded

The training completed successfully with a clear trend of decreasing loss, indicating effective model learning.

Task 2: Task 2. Sentence Embedding with Sentence BERT ✅

Implement trained BERT from task 1 with siamese network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity. (3 points) ✅

1. Use the SNLI 4 OR MNLI 5 datasets from Hugging Face, or any dataset related to classification tasks. ✅
2. Reproduce training the Sentence-BERT as described in the paper 6. ✅
3. Focus on the Classification Objective Function: (SoftmaxLoss)

o = softmax 1W T · (u, v, |u − v|)2 ✅

HINT : You can take a look how to implement Softmax loss in the file 04 - Huggingface/Appendix - Sentence Embedding/S-BERT.ipynb.

In [7]:
```
import os
import logging
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
from datasets import load_dataset, concatenate_datasets
from transformers import BertTokenizer
from tqdm.auto import tqdm
import numpy as np
from sklearn.metrics import accuracy_score, classification_report
import json
from datetime import datetime

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('sbert_training.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)
```

```
# Set device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
logger.info(f"Using device: {device}")

class SentenceBERT(nn.Module):
    def __init__(self, bert_model=None, hidden_size=256, config=None):
        super().__init__()
        if bert_model is None:
            from bert_scratch import BertModel, BertConfig
            if config is None:
                config = BertConfig()
                config.hidden_size = hidden_size
            self.bert = BertModel(config)
        else:
            self.bert = bert_model

        self.fc = nn.Linear(hidden_size * 3, 3)  # 3 classes: entailment, contra

    def mean_pooling(self, token_embeddings, attention_mask):
        input_mask_expanded = attention_mask.unsqueeze(-1).expand(token_embeddin
        return torch.sum(token_embeddings * input_mask_expanded, 1) / torch.clam

    def encode(self, input_ids, attention_mask):
        outputs = self.bert(input_ids, attention_mask=attention_mask)
        if isinstance(outputs, tuple):
            outputs = outputs[0]  # Get hidden states if tuple is returned
        embeddings = self.mean_pooling(outputs, attention_mask)
        return embeddings

    def forward(self, premise_input_ids, premise_attention_mask,
                hypothesis_input_ids, hypothesis_attention_mask):
        # Get embeddings for premise and hypothesis
        premise_embedding = self.encode(premise_input_ids, premise_attention_mas
        hypothesis_embedding = self.encode(hypothesis_input_ids, hypothesis_atte

        # Calculate cosine similarity
        cos_sim = F.cosine_similarity(premise_embedding, hypothesis_embedding)
        self.last_cos_sim = cos_sim  # Store for later use

        # Concatenate embeddings
        combined = torch.cat([
            premise_embedding,
            hypothesis_embedding,
            torch.abs(premise_embedding - hypothesis_embedding)
        ], dim=1)

        # Pass through classifier
        logits = self.fc(combined)

        return logits, cos_sim  # Return both logits and cosine similarity

def load_datasets(num_samples=None):
    logger.info("Loading SNLI and MNLI datasets...")

    # Load SNLI dataset
    snli_dataset = load_dataset("snli")

    # Load MNLI dataset
    mnli_dataset = load_dataset("multi_nli")
```

```python
    # Rename MNLI labels to match SNLI
    def rename_labels(example):
        label_map = {0: 0, 1: 1, 2: 2}  # entailment: 0, contradiction: 1, neutr
        example['label'] = label_map[example['label']]
        return example

    # Process MNLI dataset to match SNLI format
    mnli_dataset = mnli_dataset.map(rename_labels)

    # Combine datasets
    train_dataset = concatenate_datasets([
        snli_dataset['train'],
        mnli_dataset['train']
    ])

    val_dataset = concatenate_datasets([
        snli_dataset['validation'],
        mnli_dataset['validation_matched']
    ])

    # Subsample if specified
    if num_samples is not None:
        train_dataset = train_dataset.shuffle(seed=42).select(range(num_samples)
        val_dataset = val_dataset.shuffle(seed=42).select(range(num_samples))

    logger.info(f"Loaded {len(train_dataset)} training samples and {len(val_data

    return {
        'train': train_dataset,
        'validation': val_dataset
    }

def preprocess_data(datasets, tokenizer, max_length=128):
    logger.info("Preprocessing datasets...")

    def preprocess_function(examples):
        # Tokenize premises
        premise_encodings = tokenizer(
            examples['premise'],
            padding='max_length',
            truncation=True,
            max_length=max_length
        )

        # Tokenize hypotheses
        hypothesis_encodings = tokenizer(
            examples['hypothesis'],
            padding='max_length',
            truncation=True,
            max_length=max_length
        )

        return {
            'premise_input_ids': premise_encodings['input_ids'],
            'premise_attention_mask': premise_encodings['attention_mask'],
            'hypothesis_input_ids': hypothesis_encodings['input_ids'],
            'hypothesis_attention_mask': hypothesis_encodings['attention_mask'],
            'labels': examples['label']
        }
```

```python
    # Process each split
    processed_datasets = {}
    for split, dataset in datasets.items():
        processed_datasets[split] = dataset.map(
            preprocess_function,
            batched=True,
            remove_columns=dataset.column_names
        )
        processed_datasets[split].set_format('torch')

    return processed_datasets

def evaluate_model(model, dataloader):
    model.eval()
    all_predictions = []
    all_labels = []
    all_cosine_sims = []

    with torch.no_grad():
        for batch in tqdm(dataloader, desc="Evaluating"):
            # Get predictions
            outputs, cos_sim = model(
                batch['premise_input_ids'].to(device),
                batch['premise_attention_mask'].to(device),
                batch['hypothesis_input_ids'].to(device),
                batch['hypothesis_attention_mask'].to(device)
            )
            predictions = torch.argmax(outputs, dim=1)

            # Collect predictions and labels
            all_predictions.extend(predictions.cpu().numpy())
            all_labels.extend(batch['labels'].numpy())
            all_cosine_sims.extend(cos_sim.cpu().numpy())

    # Calculate metrics
    accuracy = accuracy_score(all_labels, all_predictions)
    report = classification_report(all_labels, all_predictions)

    # Calculate average cosine similarity for each class
    cosine_sims = np.array(all_cosine_sims)
    labels = np.array(all_labels)

    logger.info("\nCosine Similarity Analysis:")
    for label in [0, 1, 2]:  # entailment, contradiction, neutral
        mask = labels == label
        if mask.any():
            label_name = ['entailment', 'contradiction', 'neutral'][label]
            sims = cosine_sims[mask]
            logger.info(f"{label_name.capitalize()}: Mean={sims.mean():.4f}, Std

    return accuracy, report

def save_model(model, tokenizer, config, metrics, output_dir='sbert_model'):
    """Save the model, tokenizer, configuration and metrics."""
    os.makedirs(output_dir, exist_ok=True)

    # Save model state
    model_path = os.path.join(output_dir, 'model.pt')
    torch.save(model.state_dict(), model_path)
    logger.info(f"Model saved to {model_path}")
```

```python
    # Save tokenizer
    tokenizer_path = os.path.join(output_dir, 'tokenizer')
    tokenizer.save_pretrained(tokenizer_path)
    logger.info(f"Tokenizer saved to {tokenizer_path}")

    # Save metrics
    metrics_path = os.path.join(output_dir, 'metrics.json')
    with open(metrics_path, 'w') as f:
        json.dump(metrics, f, indent=4)
    logger.info(f"Metrics saved to {metrics_path}")

    # Save configuration
    config_path = os.path.join(output_dir, 'config.json')
    with open(config_path, 'w') as f:
        json.dump(config.__dict__, f, indent=4)
    logger.info(f"Configuration saved to {config_path}")

def main():
    # Load datasets with smaller batch size
    datasets = load_datasets(num_samples=800)
    logger.info(f"Loaded {len(datasets['train'])} training samples and {len(data

    # Initialize tokenizer and get vocab size
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    vocab_size = len(tokenizer.vocab)
    logger.info(f"Tokenizer vocabulary size: {vocab_size}")

    # Load pre-trained BERT from our custom implementation
    from bert_scratch import BertModel, BertConfig
    config = BertConfig()
    config.vocab_size = vocab_size  # Set vocab size to match tokenizer
    config.batch_size = 2  # Extremely small batch size
    config.hidden_size = 64  # Minimal hidden size
    config.num_hidden_layers = 2  # Minimal number of layers
    config.num_attention_heads = 4  # Reduced attention heads
    config.gradient_accumulation_steps = 16  # Heavy gradient accumulation
    config.intermediate_size = 256  # Minimal intermediate size
    config.max_len = 32  # Minimal sequence length
    config.attention_probs_dropout_prob = 0.1
    config.hidden_dropout_prob = 0.1

    bert_model = BertModel(config)

    # Try to load pre-trained weights if available
    try:
        checkpoints = [f for f in os.listdir('model_checkpoints') if f.startswit
        if checkpoints:
            latest_checkpoint = max(checkpoints, key=lambda x: int(x.split('_')[
            checkpoint_path = os.path.join('model_checkpoints', latest_checkpoin

            # Load checkpoint with proper handling
            checkpoint = torch.load(checkpoint_path, map_location='cpu')  # Load
            if 'model_state_dict' in checkpoint:
                state_dict = checkpoint['model_state_dict']
            else:
                state_dict = checkpoint

            # Filter out mismatched keys
            model_dict = bert_model.state_dict()
```

```
        state_dict = {k: v for k, v in state_dict.items()
                     if k in model_dict and v.shape == model_dict[k].shape}

        # Load filtered state dict
        bert_model.load_state_dict(state_dict, strict=False)
        logger.info(f"Loaded compatible weights from {checkpoint_path}")
    else:
        logger.warning("No pre-trained BERT checkpoints found. Starting with
except Exception as e:
    logger.warning(f"Failed to load pre-trained BERT weights: {e}")

# Initialize Sentence-BERT
model = SentenceBERT(bert_model, hidden_size=config.hidden_size, config=conf

# Enable gradient checkpointing for memory efficiency
if hasattr(model.bert, 'enable_gradient_checkpointing'):
    model.bert.enable_gradient_checkpointing()
    logger.info("Enabled gradient checkpointing")

# Move model to GPU after all initialization
model = model.to(device)

# Clear GPU memory before starting
torch.cuda.empty_cache()

# Preprocess datasets with reduced sequence length
tokenized_datasets = preprocess_data(datasets, tokenizer, max_length=config.

# Create dataloaders with minimal batch size
train_dataloader = DataLoader(
    tokenized_datasets['train'],
    batch_size=config.batch_size,
    shuffle=True,
    pin_memory=True,
    num_workers=0,
    persistent_workers=False
)

val_dataloader = DataLoader(
    tokenized_datasets['validation'],
    batch_size=config.batch_size,
    pin_memory=True,
    num_workers=0,
    persistent_workers=False
)

# Initialize optimizer with weight decay
no_decay = ['bias', 'LayerNorm.weight']
optimizer_grouped_parameters = [
    {
        'params': [p for n, p in model.named_parameters() if not any(nd in n
        'weight_decay': config.weight_decay
    },
    {
        'params': [p for n, p in model.named_parameters() if any(nd in n for
        'weight_decay': 0.0
    }
]

optimizer = optim.AdamW(
```

```
        optimizer_grouped_parameters,
        lr=config.learning_rate,
        eps=config.adam_epsilon
    )

    # Training Loop
    num_epochs = 10
    criterion = nn.CrossEntropyLoss()
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10)

    try:
        for epoch in range(num_epochs):
            model.train()
            total_loss = 0
            epoch_cos_sims = []

            progress_bar = tqdm(train_dataloader, desc=f"Epoch {epoch+1}")
            for step, batch in enumerate(progress_bar):
                # Zero gradients
                optimizer.zero_grad()

                # Forward pass
                outputs, cos_sim = model(
                    batch['premise_input_ids'].to(device),
                    batch['premise_attention_mask'].to(device),
                    batch['hypothesis_input_ids'].to(device),
                    batch['hypothesis_attention_mask'].to(device)
                )

                # Calculate loss
                loss = criterion(outputs, batch['labels'].to(device))

                # Backward pass
                loss.backward()

                # Clip gradients
                torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

                # Update parameters
                optimizer.step()
                scheduler.step()

                # Update metrics
                total_loss += loss.item()
                epoch_cos_sims.extend(cos_sim.detach().cpu().numpy())

                # Update progress bar
                progress_bar.set_postfix({'loss': total_loss / (step + 1)})

                # Cleanup
                del outputs, loss, batch
                torch.cuda.empty_cache()

            # Calculate epoch metrics
            avg_loss = total_loss / len(train_dataloader)
            epoch_cos_sims = np.array(epoch_cos_sims)

            # Print epoch summary
            logger.info(f"Epoch {epoch+1} completed. Average loss: {avg_loss:.4f
            logger.info(f"Epoch {epoch+1} Cosine Similarities - Mean: {epoch_cos
```

```python
            # Evaluate
            accuracy, report = evaluate_model(model, val_dataloader)
            logger.info(f"Validation metrics: ({accuracy}, '{report}')")

            # Save model
            metrics_dict = {
                'accuracy': float(accuracy),
                'loss': float(avg_loss),
                'classification_report': report,
                'cosine_similarity_mean': float(epoch_cos_sims.mean()),
                'cosine_similarity_std': float(epoch_cos_sims.std())
            }
            save_model(model, tokenizer, config, metrics_dict, output_dir='sbert

    except KeyboardInterrupt:
        logger.info("Training interrupted by user")
    except Exception as e:
        logger.error(f"Error during training: {e}")
        raise
    finally:
        logger.info("Training completed!")

if __name__ == "__main__":
    main()
```

```
2025-02-17 17:30:52,384 - INFO - Using device: cuda
2025-02-17 17:30:52,391 - INFO - Loading SNLI and MNLI datasets...
2025-02-17 17:31:08,866 - INFO - Loaded 800 training samples and 800 validation s
amples
2025-02-17 17:31:08,868 - INFO - Loaded 800 training samples and 800 validation s
amples
2025-02-17 17:31:09,210 - INFO - Tokenizer vocabulary size: 30522
/tmp/ipykernel_1859902/3502560219.py:259: FutureWarning: You are using `torch.loa
d` with `weights_only=False` (the current default value), which uses the default
pickle module implicitly. It is possible to construct malicious pickle data which
will execute arbitrary code during unpickling (See https://github.com/pytorch/pyt
orch/blob/main/SECURITY.md#untrusted-models for more details). In a future releas
e, the default value for `weights_only` will be flipped to `True`. This limits th
e functions that could be executed during unpickling. Arbitrary objects will no l
onger be allowed to be loaded via this mode unless they are explicitly allowliste
d by the user via `torch.serialization.add_safe_globals`. We recommend you start
setting `weights_only=True` for any use case where you don't have full control of
the loaded file. Please open an issue on GitHub for any issues related to this ex
perimental feature.
  checkpoint = torch.load(checkpoint_path, map_location='cpu')  # Load to CPU fir
st
2025-02-17 17:31:09,327 - INFO - Loaded compatible weights from model_checkpoint
s/bert_epoch_15_20250217_172515.pt
2025-02-17 17:31:09,329 - INFO - Enabled gradient checkpointing
2025-02-17 17:31:09,417 - INFO - Preprocessing datasets...
Epoch 1:   0%|          | 0/400 [00:00<?, ?it/s]
2025-02-17 17:31:31,000 - INFO - Epoch 1 completed. Average loss: 1.1087
2025-02-17 17:31:31,002 - INFO - Epoch 1 Cosine Similarities - Mean: 0.9317, Std:
0.0296
Evaluating:   0%|          | 0/400 [00:00<?, ?it/s]
```

```
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
2025-02-17 17:31:34,696 - INFO - 
Cosine Similarity Analysis:
2025-02-17 17:31:34,697 - INFO - Entailment: Mean=0.9508, Std=0.0280
2025-02-17 17:31:34,698 - INFO - Contradiction: Mean=0.9532, Std=0.0235
2025-02-17 17:31:34,699 - INFO - Neutral: Mean=0.9504, Std=0.0254
2025-02-17 17:31:34,700 - INFO - Validation metrics: (0.35375, '              pre
cision    recall  f1-score   support

          -1       0.00      0.00      0.00         8
           0       0.35      0.56      0.43       273
           1       0.36      0.47      0.41       273
           2       0.50      0.00      0.01       246

    accuracy                           0.35       800
   macro avg       0.30      0.26      0.21       800
weighted avg       0.40      0.35      0.29       800
')
2025-02-17 17:31:34,729 - INFO - Model saved to sbert_model/model.pt
2025-02-17 17:31:34,749 - INFO - Tokenizer saved to sbert_model/tokenizer
2025-02-17 17:31:34,750 - INFO - Metrics saved to sbert_model/metrics.json
2025-02-17 17:31:34,751 - INFO - Configuration saved to sbert_model/config.json
Epoch 2:   0%|          | 0/400 [00:00<?, ?it/s]
/home/jupyter-st125462/.local/lib/python3.12/site-packages/torch/_dynamo/eval_fra
me.py:632: UserWarning: torch.utils.checkpoint: the use_reentrant parameter shoul
d be passed explicitly. In version 2.5 we will raise an exception if use_reentran
t is not passed. use_reentrant=False is recommended, but if you need to preserve
the current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
2025-02-17 17:31:55,485 - INFO - Epoch 2 completed. Average loss: 1.1006
2025-02-17 17:31:55,487 - INFO - Epoch 2 Cosine Similarities - Mean: 0.9296, Std:
0.0310
Evaluating:   0%|          | 0/400 [00:00<?, ?it/s]
```

```
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
2025-02-17 17:31:59,168 - INFO - 
Cosine Similarity Analysis:
2025-02-17 17:31:59,169 - INFO - Entailment: Mean=0.9464, Std=0.0309
2025-02-17 17:31:59,170 - INFO - Contradiction: Mean=0.9493, Std=0.0261
2025-02-17 17:31:59,171 - INFO - Neutral: Mean=0.9463, Std=0.0280
2025-02-17 17:31:59,172 - INFO - Validation metrics: (0.3275, '              prec
ision    recall  f1-score   support

          -1       0.00      0.00      0.00         8
           0       0.35      0.66      0.46       273
           1       0.36      0.09      0.14       273
           2       0.27      0.24      0.25       246

    accuracy                           0.33       800
   macro avg       0.24      0.25      0.21       800
weighted avg       0.32      0.33      0.28       800
')
2025-02-17 17:31:59,657 - INFO - Model saved to sbert_model/model.pt
2025-02-17 17:31:59,700 - INFO - Tokenizer saved to sbert_model/tokenizer
2025-02-17 17:31:59,702 - INFO - Metrics saved to sbert_model/metrics.json
2025-02-17 17:31:59,703 - INFO - Configuration saved to sbert_model/config.json
Epoch 3:   0%|          | 0/400 [00:00<?, ?it/s]
/home/jupyter-st125462/.local/lib/python3.12/site-packages/torch/_dynamo/eval_fra
me.py:632: UserWarning: torch.utils.checkpoint: the use_reentrant parameter shoul
d be passed explicitly. In version 2.5 we will raise an exception if use_reentran
t is not passed. use_reentrant=False is recommended, but if you need to preserve
the current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
2025-02-17 17:32:20,828 - INFO - Epoch 3 completed. Average loss: 1.0883
2025-02-17 17:32:20,830 - INFO - Epoch 3 Cosine Similarities - Mean: 0.9236, Std:
0.0361
Evaluating:   0%|          | 0/400 [00:00<?, ?it/s]
```

```
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
2025-02-17 17:32:24,525 - INFO -
Cosine Similarity Analysis:
2025-02-17 17:32:24,526 - INFO - Entailment: Mean=0.9404, Std=0.0356
2025-02-17 17:32:24,527 - INFO - Contradiction: Mean=0.9442, Std=0.0299
2025-02-17 17:32:24,528 - INFO - Neutral: Mean=0.9409, Std=0.0320
2025-02-17 17:32:24,529 - INFO - Validation metrics: (0.3475, '              prec
ision    recall  f1-score   support

          -1       0.00      0.00      0.00         8
           0       0.42      0.19      0.27       273
           1       0.36      0.67      0.47       273
           2       0.25      0.17      0.21       246

    accuracy                           0.35       800
   macro avg       0.26      0.26      0.24       800
weighted avg       0.34      0.35      0.31       800
')
2025-02-17 17:32:24,923 - INFO - Model saved to sbert_model/model.pt
2025-02-17 17:32:24,946 - INFO - Tokenizer saved to sbert_model/tokenizer
2025-02-17 17:32:24,948 - INFO - Metrics saved to sbert_model/metrics.json
2025-02-17 17:32:24,949 - INFO - Configuration saved to sbert_model/config.json
Epoch 4:   0%|          | 0/400 [00:00<?, ?it/s]
/home/jupyter-st125462/.local/lib/python3.12/site-packages/torch/_dynamo/eval_fra
me.py:632: UserWarning: torch.utils.checkpoint: the use_reentrant parameter shoul
d be passed explicitly. In version 2.5 we will raise an exception if use_reentran
t is not passed. use_reentrant=False is recommended, but if you need to preserve
the current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
2025-02-17 17:32:45,836 - INFO - Epoch 4 completed. Average loss: 1.0782
2025-02-17 17:32:45,837 - INFO - Epoch 4 Cosine Similarities - Mean: 0.9149, Std:
0.0411
Evaluating:   0%|          | 0/400 [00:00<?, ?it/s]
```

```
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
2025-02-17 17:32:49,531 - INFO -
Cosine Similarity Analysis:
2025-02-17 17:32:49,533 - INFO - Entailment: Mean=0.9309, Std=0.0434
2025-02-17 17:32:49,533 - INFO - Contradiction: Mean=0.9364, Std=0.0364
2025-02-17 17:32:49,535 - INFO - Neutral: Mean=0.9325, Std=0.0387
2025-02-17 17:32:49,536 - INFO - Validation metrics: (0.3675, '              prec
ision    recall  f1-score   support

          -1       0.00      0.00      0.00         8
           0       0.45      0.12      0.19       273
           1       0.36      0.94      0.52       273
           2       0.36      0.02      0.04       246

    accuracy                           0.37       800
   macro avg       0.29      0.27      0.19       800
weighted avg       0.38      0.37      0.25       800
')
2025-02-17 17:32:50,085 - INFO - Model saved to sbert_model/model.pt
2025-02-17 17:32:50,123 - INFO - Tokenizer saved to sbert_model/tokenizer
2025-02-17 17:32:50,125 - INFO - Metrics saved to sbert_model/metrics.json
2025-02-17 17:32:50,126 - INFO - Configuration saved to sbert_model/config.json
Epoch 5:   0%|          | 0/400 [00:00<?, ?it/s]
/home/jupyter-st125462/.local/lib/python3.12/site-packages/torch/_dynamo/eval_fra
me.py:632: UserWarning: torch.utils.checkpoint: the use_reentrant parameter shoul
d be passed explicitly. In version 2.5 we will raise an exception if use_reentran
t is not passed. use_reentrant=False is recommended, but if you need to preserve
the current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
2025-02-17 17:33:10,826 - INFO - Epoch 5 completed. Average loss: 1.0695
2025-02-17 17:33:10,828 - INFO - Epoch 5 Cosine Similarities - Mean: 0.9006, Std:
0.0498
Evaluating:   0%|          | 0/400 [00:00<?, ?it/s]
```

```
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
2025-02-17 17:33:14,524 - INFO -
Cosine Similarity Analysis:
2025-02-17 17:33:14,525 - INFO - Entailment: Mean=0.9173, Std=0.0536
2025-02-17 17:33:14,527 - INFO - Contradiction: Mean=0.9243, Std=0.0455
2025-02-17 17:33:14,527 - INFO - Neutral: Mean=0.9202, Std=0.0470
2025-02-17 17:33:14,528 - INFO - Validation metrics: (0.38375, '              pre
cision    recall  f1-score   support

          -1       0.00      0.00      0.00         8
           0       0.45      0.30      0.36       273
           1       0.37      0.81      0.51       273
           2       0.27      0.01      0.02       246

    accuracy                           0.38       800
   macro avg       0.27      0.28      0.22       800
weighted avg       0.36      0.38      0.30       800
')
2025-02-17 17:33:14,972 - INFO - Model saved to sbert_model/model.pt
2025-02-17 17:33:15,013 - INFO - Tokenizer saved to sbert_model/tokenizer
2025-02-17 17:33:15,014 - INFO - Metrics saved to sbert_model/metrics.json
2025-02-17 17:33:15,016 - INFO - Configuration saved to sbert_model/config.json
Epoch 6:   0%|          | 0/400 [00:00<?, ?it/s]
/home/jupyter-st125462/.local/lib/python3.12/site-packages/torch/_dynamo/eval_fra
me.py:632: UserWarning: torch.utils.checkpoint: the use_reentrant parameter shoul
d be passed explicitly. In version 2.5 we will raise an exception if use_reentran
t is not passed. use_reentrant=False is recommended, but if you need to preserve
the current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
2025-02-17 17:33:35,726 - INFO - Epoch 6 completed. Average loss: 1.0682
2025-02-17 17:33:35,727 - INFO - Epoch 6 Cosine Similarities - Mean: 0.8822, Std:
0.0628
Evaluating:   0%|          | 0/400 [00:00<?, ?it/s]
```

```
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
2025-02-17 17:33:39,390 - INFO -
Cosine Similarity Analysis:
2025-02-17 17:33:39,391 - INFO - Entailment: Mean=0.8948, Std=0.0722
2025-02-17 17:33:39,391 - INFO - Contradiction: Mean=0.9052, Std=0.0614
2025-02-17 17:33:39,392 - INFO - Neutral: Mean=0.9004, Std=0.0620
2025-02-17 17:33:39,393 - INFO - Validation metrics: (0.39625, '              pre
cision    recall  f1-score   support

          -1       0.00      0.00      0.00         8
           0       0.42      0.40      0.41       273
           1       0.40      0.66      0.50       273
           2       0.31      0.11      0.16       246

    accuracy                           0.40       800
   macro avg       0.28      0.29      0.27       800
weighted avg       0.37      0.40      0.36       800
')
2025-02-17 17:33:39,730 - INFO - Model saved to sbert_model/model.pt
2025-02-17 17:33:39,769 - INFO - Tokenizer saved to sbert_model/tokenizer
2025-02-17 17:33:39,771 - INFO - Metrics saved to sbert_model/metrics.json
2025-02-17 17:33:39,772 - INFO - Configuration saved to sbert_model/config.json
Epoch 7:   0%|          | 0/400 [00:00<?, ?it/s]
/home/jupyter-st125462/.local/lib/python3.12/site-packages/torch/_dynamo/eval_fra
me.py:632: UserWarning: torch.utils.checkpoint: the use_reentrant parameter shoul
d be passed explicitly. In version 2.5 we will raise an exception if use_reentran
t is not passed. use_reentrant=False is recommended, but if you need to preserve
the current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
2025-02-17 17:34:00,095 - INFO - Epoch 7 completed. Average loss: 1.0524
2025-02-17 17:34:00,096 - INFO - Epoch 7 Cosine Similarities - Mean: 0.8602, Std:
0.0784
Evaluating:   0%|          | 0/400 [00:00<?, ?it/s]
```

```
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
2025-02-17 17:34:02,914 - INFO - 
Cosine Similarity Analysis:
2025-02-17 17:34:02,915 - INFO - Entailment: Mean=0.8690, Std=0.0957
2025-02-17 17:34:02,917 - INFO - Contradiction: Mean=0.8843, Std=0.0812
2025-02-17 17:34:02,917 - INFO - Neutral: Mean=0.8785, Std=0.0808
2025-02-17 17:34:02,918 - INFO - Validation metrics: (0.3675, '              prec
ision    recall  f1-score   support

          -1       0.00      0.00      0.00         8
           0       0.44      0.33      0.38       273
           1       0.40      0.31      0.35       273
           2       0.31      0.48      0.38       246

    accuracy                           0.37       800
   macro avg       0.29      0.28      0.28       800
weighted avg       0.38      0.37      0.36       800
')
2025-02-17 17:34:03,304 - INFO - Model saved to sbert_model/model.pt
2025-02-17 17:34:03,342 - INFO - Tokenizer saved to sbert_model/tokenizer
2025-02-17 17:34:03,344 - INFO - Metrics saved to sbert_model/metrics.json
2025-02-17 17:34:03,345 - INFO - Configuration saved to sbert_model/config.json
Epoch 8:   0%|          | 0/400 [00:00<?, ?it/s]
/home/jupyter-st125462/.local/lib/python3.12/site-packages/torch/_dynamo/eval_fra
me.py:632: UserWarning: torch.utils.checkpoint: the use_reentrant parameter shoul
d be passed explicitly. In version 2.5 we will raise an exception if use_reentran
t is not passed. use_reentrant=False is recommended, but if you need to preserve
the current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
2025-02-17 17:34:24,456 - INFO - Epoch 8 completed. Average loss: 1.0448
2025-02-17 17:34:24,457 - INFO - Epoch 8 Cosine Similarities - Mean: 0.8389, Std:
0.0938
Evaluating:   0%|          | 0/400 [00:00<?, ?it/s]
```

```
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
2025-02-17 17:34:28,143 - INFO - 
Cosine Similarity Analysis:
2025-02-17 17:34:28,144 - INFO - Entailment: Mean=0.8479, Std=0.1113
2025-02-17 17:34:28,145 - INFO - Contradiction: Mean=0.8646, Std=0.0956
2025-02-17 17:34:28,146 - INFO - Neutral: Mean=0.8588, Std=0.0933
2025-02-17 17:34:28,147 - INFO - Validation metrics: (0.4, '              precisi
on    recall  f1-score   support

          -1       0.00      0.00      0.00         8
           0       0.43      0.44      0.43       273
           1       0.42      0.48      0.45       273
           2       0.33      0.28      0.31       246

    accuracy                           0.40       800
   macro avg       0.30      0.30      0.30       800
weighted avg       0.39      0.40      0.39       800
')
2025-02-17 17:34:28,255 - INFO - Model saved to sbert_model/model.pt
2025-02-17 17:34:28,290 - INFO - Tokenizer saved to sbert_model/tokenizer
2025-02-17 17:34:28,292 - INFO - Metrics saved to sbert_model/metrics.json
2025-02-17 17:34:28,293 - INFO - Configuration saved to sbert_model/config.json
Epoch 9:   0%|          | 0/400 [00:00<?, ?it/s]
/home/jupyter-st125462/.local/lib/python3.12/site-packages/torch/_dynamo/eval_fra
me.py:632: UserWarning: torch.utils.checkpoint: the use_reentrant parameter shoul
d be passed explicitly. In version 2.5 we will raise an exception if use_reentran
t is not passed. use_reentrant=False is recommended, but if you need to preserve
the current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
2025-02-17 17:34:48,948 - INFO - Epoch 9 completed. Average loss: 1.0398
2025-02-17 17:34:48,949 - INFO - Epoch 9 Cosine Similarities - Mean: 0.8044, Std:
0.1208
Evaluating:   0%|          | 0/400 [00:00<?, ?it/s]
```

```
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
2025-02-17 17:34:51,855 - INFO -
Cosine Similarity Analysis:
2025-02-17 17:34:51,856 - INFO - Entailment: Mean=0.8104, Std=0.1461
2025-02-17 17:34:51,857 - INFO - Contradiction: Mean=0.8331, Std=0.1255
2025-02-17 17:34:51,858 - INFO - Neutral: Mean=0.8260, Std=0.1206
2025-02-17 17:34:51,858 - INFO - Validation metrics: (0.40375, '              pre
cision    recall  f1-score   support

          -1       0.00      0.00      0.00         8
           0       0.43      0.45      0.44       273
           1       0.43      0.51      0.47       273
           2       0.32      0.25      0.28       246

    accuracy                           0.40       800
   macro avg       0.30      0.30      0.30       800
weighted avg       0.39      0.40      0.40       800
')
2025-02-17 17:34:51,966 - INFO - Model saved to sbert_model/model.pt
2025-02-17 17:34:52,000 - INFO - Tokenizer saved to sbert_model/tokenizer
2025-02-17 17:34:52,002 - INFO - Metrics saved to sbert_model/metrics.json
2025-02-17 17:34:52,003 - INFO - Configuration saved to sbert_model/config.json
Epoch 10:   0%|          | 0/400 [00:00<?, ?it/s]
/home/jupyter-st125462/.local/lib/python3.12/site-packages/torch/_dynamo/eval_fra
me.py:632: UserWarning: torch.utils.checkpoint: the use_reentrant parameter shoul
d be passed explicitly. In version 2.5 we will raise an exception if use_reentran
t is not passed. use_reentrant=False is recommended, but if you need to preserve
the current default behavior, you can pass use_reentrant=True. Refer to docs for
more details on the differences between the two variants.
  return fn(*args, **kwargs)
2025-02-17 17:35:12,721 - INFO - Epoch 10 completed. Average loss: 1.0340
2025-02-17 17:35:12,722 - INFO - Epoch 10 Cosine Similarities - Mean: 0.7670, St
d: 0.1534
Evaluating:   0%|          | 0/400 [00:00<?, ?it/s]
```

```
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/home/jupyter-st125462/.local/lib/python3.12/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
2025-02-17 17:35:16,390 - INFO -
Cosine Similarity Analysis:
2025-02-17 17:35:16,391 - INFO - Entailment: Mean=0.7780, Std=0.1795
2025-02-17 17:35:16,392 - INFO - Contradiction: Mean=0.8076, Std=0.1534
2025-02-17 17:35:16,393 - INFO - Neutral: Mean=0.7976, Std=0.1491
2025-02-17 17:35:16,394 - INFO - Validation metrics: (0.4025, '              prec
ision    recall  f1-score   support

          -1       0.00      0.00      0.00         8
           0       0.43      0.38      0.41       273
           1       0.41      0.61      0.49       273
           2       0.33      0.21      0.25       246

    accuracy                           0.40       800
   macro avg       0.29      0.30      0.29       800
weighted avg       0.39      0.40      0.38       800
')
2025-02-17 17:35:16,481 - INFO - Model saved to sbert_model/model.pt
2025-02-17 17:35:16,500 - INFO - Tokenizer saved to sbert_model/tokenizer
2025-02-17 17:35:16,501 - INFO - Metrics saved to sbert_model/metrics.json
2025-02-17 17:35:16,503 - INFO - Configuration saved to sbert_model/config.json
2025-02-17 17:35:16,503 - INFO - Training completed!
```

Findings

BERT Pre-training Results

- Training completed 15 epochs
- Initial vocabulary size: 27,092 tokens
- Training device: GPU 1 with 11GB free memory
- Loss progression:
  - Starting loss: 7.9539 (Epoch 1)
  - Final loss: 5.0737 (Epoch 15)
  - Overall loss reduction: ~36%

SBERT Fine-tuning Results

- Dataset: Combined SNLI and MNLI (800 training + 800 validation samples)
- Model configuration:

- Vocabulary size: 30,522
  - Hidden size: 64
  - Layers: 2
  - Attention heads: 4

Final Performance Metrics (After 10 epochs)

Metric	Value
Accuracy	40.25%
Average Loss	1.0340
Cosine Similarity Mean	0.7670
Cosine Similarity Std	0.1534

Class-wise Performance

Class	Precision	Recall	F1-Score
Entailment	0.43	0.38	0.41
Contradiction	0.41	0.61	0.49
Neutral	0.33	0.21	0.25

Training Progression

- Started with high cosine similarities (~0.93)
- Gradually decreased to more discriminative values (~0.77)
- Model showed steady improvement in classification performance
- Final weighted average metrics:
  - Precision: 0.39
  - Recall: 0.40
  - F1-score: 0.38

Both training sessions completed successfully with proper model and checkpoint saving at each epoch.

Task 3. Evaluation and Analysis (1 points) ✅

1. Provide the performance metrics based on the SNLI or MNLI datasets for the Natural Language Inference (NLI) task. ✅
2. Discuss any limitations or challenges encountered during the implementation and propose potential improvements or modifications. ✅

NOTE: Make sure to provide proper documentation, including details of the datasets used, hyperparameters, and any modifications made to the original models

---

1. Performance Metrics

Table 1. Performance Table (SNLI + MNLI Combined Dataset)

Metric	Value
Overall Accuracy	40.25%
Average Loss	1.0340
Macro Avg F1-score	0.29
Weighted Avg F1-score	0.38

Class-wise Performance

Class	Precision	Recall	F1-Score
Entailment	0.43	0.38	0.41
Contradiction	0.41	0.61	0.49
Neutral	0.33	0.21	0.25

Cosine Similarity Analysis (Final Epoch)

Class	Mean	Std
Entailment	0.7780	0.1795
Contradiction	0.8076	0.1534
Neutral	0.7976	0.1491

2. Implementation Details

Dataset Information

- Combined SNLI and MNLI datasets
- Training samples: 800
- Validation samples: 800
- Tokenizer vocabulary size: 30,522

Hyperparameters

- Hidden size: 64
- Number of layers: 2

- Attention heads: 4
- Batch size: 2
- Sequence length: 32
- Learning rate: Not explicitly stated in logs
- Training epochs: 10
- Gradient accumulation steps: 16

3. Limitations and Challenges

1. **Resource Constraints:**

   - Had to use minimal model architecture due to memory limitations
   - Required gradient checkpointing for memory efficiency
   - Small batch size (2) needed to fit in memory
2. **Performance Limitations:**

   - Relatively low accuracy (40.25%)
   - Poor performance on neutral class (F1: 0.25)
   - High cosine similarities between different classes
3. **Dataset Challenges:**

   - Small training set (800 samples) may not be representative
   - Imbalanced class distribution
   - Some invalid labels present (-1 class with 8 samples)

4. Proposed Improvements

1. **Model Architecture:**

   - Increase model capacity (more layers, wider hidden dimensions)
   - Implement attention mechanisms specific to NLI tasks
   - Add residual connections for better gradient flow
2. **Training Strategy:**

   - Use larger batch sizes with gradient accumulation
   - Implement curriculum learning
   - Add contrastive learning objectives
3. **Data Processing:**

   - Use larger training dataset
   - Balance class distribution
   - Better handling of invalid labels
   - Implement data augmentation techniques
4. **Optimization:**

   - Try different learning rate schedules
   - Implement early stopping
   - Use mixed-precision training

- Add regularization techniques

The implementation shows proof of concept but would benefit from these improvements for production use.

Task 4. Text similarity - Web Application Development ✅

Develop a simple web application that demonstrates the capabilities of your text-embedding model. (1 points) ✅

1. Develop a simple website with two input boxes for search queries. ✅
2. Utilize a custom-trained sentence transformer model to predict Natural Language Inference (NLI) ✅

Task (entailment, neutral and contradiction). For example: • Premise: A man is playing a guitar on stage. • Hypothesis: The man is performing music. • Label: Entailment

The app is available on GitHub Lionks mentioned in the beginning of the notebook!

Below is the inference script for the same:

In [3]:
```
import torch
import json
from transformers import BertTokenizer
from sentence_bert import SentenceBERT
from bert_scratch import BertConfig
import logging

# Configure logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

class NLIPredictor:
    def __init__(self, model_dir='sbert_model'):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu
        logger.info(f"Using device: {self.device}")

        # Initialize tokenizer first to get vocab size
        self.tokenizer = BertTokenizer.from_pretrained(f"{model_dir}/tokenizer")
        logger.info(f"Loaded tokenizer with vocabulary size: {len(self.tokenizer

        # Initialize BERT config with SBERT values
        from bert_scratch import BertConfig
        config = BertConfig()

        # Set model architecture parameters
        config.vocab_size = 30522  # Fixed vocab size from SBERT config
        config.hidden_size = 64
        config.num_hidden_layers = 2
        config.num_attention_heads = 4
```

```
        config.intermediate_size = 256
        config.max_position_embeddings = 128
        config.max_len = 32
        config.type_vocab_size = 2

        # Set dropout and normalization parameters
        config.hidden_dropout_prob = 0.1
        config.attention_probs_dropout_prob = 0.1
        config.layer_norm_eps = 1e-12

        # Set special token IDs
        config.pad_token_id = 0
        config.mask_token_id = 3
        config.cls_token_id = 1
        config.sep_token_id = 2

        # Training parameters (required by BertConfig)
        config.learning_rate = 1e-4
        config.batch_size = 2
        config.gradient_accumulation_steps = 16
        config.weight_decay = 0.01
        config.adam_epsilon = 1e-8
        config.warmup_ratio = 0.1

        # Initialize SentenceBERT with the config
        from sentence_bert import SentenceBERT
        self.model = SentenceBERT(None, hidden_size=config.hidden_size, config=c

        # Load model weights
        try:
            model_path = f"{model_dir}/model.pt"
            state_dict = torch.load(model_path, map_location=self.device)
            self.model.load_state_dict(state_dict)
            logger.info(f"Loaded model weights from {model_path}")
        except Exception as e:
            logger.error(f"Error loading model weights: {e}")
            raise

        self.model.to(self.device)
        self.model.eval()

        # Label mapping
        self.id2label = {0: 'entailment', 1: 'contradiction', 2: 'neutral'}

        logger.info("Model initialized successfully")

    def predict(self, premise, hypothesis):
        # Clear GPU cache before prediction
        if torch.cuda.is_available():
            torch.cuda.empty_cache()

        # Tokenize inputs
        inputs = self.tokenizer(
            [premise, hypothesis],
            padding=True,
            truncation=True,
            max_length=32,  # Use fixed max_length from config
            return_tensors='pt'
        )
```

```
        # Split inputs for premise and hypothesis
        premise_input_ids = inputs['input_ids'][0].unsqueeze(0)
        premise_attention_mask = inputs['attention_mask'][0].unsqueeze(0)
        hypothesis_input_ids = inputs['input_ids'][1].unsqueeze(0)
        hypothesis_attention_mask = inputs['attention_mask'][1].unsqueeze(0)

        # Move to device
        premise_input_ids = premise_input_ids.to(self.device)
        premise_attention_mask = premise_attention_mask.to(self.device)
        hypothesis_input_ids = hypothesis_input_ids.to(self.device)
        hypothesis_attention_mask = hypothesis_attention_mask.to(self.device)

        # Get prediction
        with torch.no_grad():
            with torch.amp.autocast(device_type='cuda', dtype=torch.float16):
                outputs = self.model(
                    premise_input_ids,
                    premise_attention_mask,
                    hypothesis_input_ids,
                    hypothesis_attention_mask
                )
                prediction = torch.argmax(outputs, dim=1).item()
                probabilities = torch.nn.functional.softmax(outputs, dim=1)[0]

        result = {
            'label': self.id2label[prediction],
            'probabilities': {
                self.id2label[i]: float(prob.item())
                for i, prob in enumerate(probabilities)
            }
        }

        # Clear memory
        if torch.cuda.is_available():
            torch.cuda.empty_cache()

        return result

def main():
    # Initialize predictor
    predictor = NLIPredictor()

    # Example usage
    examples = [
        {
            'premise': 'A man is playing a guitar on stage.',
            'hypothesis': 'The man is performing music.',
            'expected': 'entailment'
        },
        {
            'premise': 'The cat is sleeping on the couch.',
            'hypothesis': 'The dog is running in the park.',
            'expected': 'contradiction'
        },
        {
            'premise': 'A woman is reading a book.',
            'hypothesis': 'She is wearing glasses.',
            'expected': 'neutral'
        },
        {
```

```
    'premise': 'Children are playing soccer in the park.',
    'hypothesis': 'Kids are engaged in outdoor sports.',
    'expected': 'entailment'
},
{
    'premise': 'The restaurant is packed with customers.',
    'hypothesis': 'The restaurant is closed today.',
    'expected': 'contradiction'
},
{
    'premise': 'A student is writing notes in class.',
    'hypothesis': 'The student understands the material.',
    'expected': 'neutral'
},
{
    'premise': 'The chef is preparing pasta in the kitchen.',
    'hypothesis': 'Someone is cooking food.',
    'expected': 'entailment'
},
{
    'premise': 'The sky is clear and blue today.',
    'hypothesis': 'It is raining heavily.',
    'expected': 'contradiction'
},
{
    'premise': 'A person bought a new laptop.',
    'hypothesis': 'They got it from Amazon.',
    'expected': 'neutral'
},
{
    'premise': 'The train arrived 30 minutes late.',
    'hypothesis': 'The train was delayed.',
    'expected': 'entailment'
},
{
    'premise': 'The museum is open on weekends.',
    'hypothesis': 'The museum is closed every day.',
    'expected': 'contradiction'
},
{
    'premise': 'A woman is walking her dog.',
    'hypothesis': 'The dog is brown in color.',
    'expected': 'neutral'
},
{
    'premise': 'The movie theater is showing new releases.',
    'hypothesis': 'Films are being screened.',
    'expected': 'entailment'
}
]

# Test each example
for i, example in enumerate(examples, 1):
    result = predictor.predict(example['premise'], example['hypothesis'])
    logger.info(f"\nExample {i}:")
    logger.info(f"Premise: {example['premise']}")
    logger.info(f"Hypothesis: {example['hypothesis']}")
    logger.info(f"Expected: {example['expected']}")
    logger.info(f"Predicted: {result['label']}")
    logger.info("Probabilities:")
```

```
        for label, prob in result['probabilities'].items():
            logger.info(f"  {label}: {prob:.3f}")

if __name__ == "__main__":
    main()
```

```
2025-02-22 08:43:49,069 - INFO - Using device: cuda
2025-02-22 08:43:49,120 - INFO - Selected GPU 0 with 11004.50MB free memory
2025-02-22 08:43:49,121 - INFO - Using device: cuda:0
2025-02-22 08:43:49,123 - INFO - Using device: cuda
2025-02-22 08:43:49,165 - INFO - Loaded tokenizer with vocabulary size: 30522
/tmp/ipykernel_2166839/3675333752.py:64: FutureWarning: You are using `torch.load
` with `weights_only=False` (the current default value), which uses the default p
ickle module implicitly. It is possible to construct malicious pickle data which
will execute arbitrary code during unpickling (See https://github.com/pytorch/pyt
orch/blob/main/SECURITY.md#untrusted-models for more details). In a future releas
e, the default value for `weights_only` will be flipped to `True`. This limits th
e functions that could be executed during unpickling. Arbitrary objects will no l
onger be allowed to be loaded via this mode unless they are explicitly allowliste
d by the user via `torch.serialization.add_safe_globals`. We recommend you start
setting `weights_only=True` for any use case where you don't have full control of
the loaded file. Please open an issue on GitHub for any issues related to this ex
perimental feature.
  state_dict = torch.load(model_path, map_location=self.device)
2025-02-22 08:43:49,432 - INFO - Loaded model weights from sbert_model/model.pt
2025-02-22 08:43:49,439 - INFO - Model initialized successfully
2025-02-22 08:43:49,830 - INFO - 
Example 1:
2025-02-22 08:43:49,831 - INFO - Premise: A man is playing a guitar on stage.
2025-02-22 08:43:49,832 - INFO - Hypothesis: The man is performing music.
2025-02-22 08:43:49,832 - INFO - Expected: entailment
2025-02-22 08:43:49,833 - INFO - Predicted: contradiction
2025-02-22 08:43:49,834 - INFO - Probabilities:
2025-02-22 08:43:49,835 - INFO -   entailment: 0.301
2025-02-22 08:43:49,835 - INFO -   contradiction: 0.389
2025-02-22 08:43:49,836 - INFO -   neutral: 0.310
2025-02-22 08:43:49,846 - INFO - 
Example 2:
2025-02-22 08:43:49,847 - INFO - Premise: The cat is sleeping on the couch.
2025-02-22 08:43:49,847 - INFO - Hypothesis: The dog is running in the park.
2025-02-22 08:43:49,848 - INFO - Expected: contradiction
2025-02-22 08:43:49,849 - INFO - Predicted: neutral
2025-02-22 08:43:49,849 - INFO - Probabilities:
2025-02-22 08:43:49,850 - INFO -   entailment: 0.216
2025-02-22 08:43:49,851 - INFO -   contradiction: 0.388
2025-02-22 08:43:49,851 - INFO -   neutral: 0.396
2025-02-22 08:43:49,861 - INFO - 
Example 3:
2025-02-22 08:43:49,861 - INFO - Premise: A woman is reading a book.
2025-02-22 08:43:49,862 - INFO - Hypothesis: She is wearing glasses.
2025-02-22 08:43:49,862 - INFO - Expected: neutral
2025-02-22 08:43:49,863 - INFO - Predicted: entailment
2025-02-22 08:43:49,864 - INFO - Probabilities:
2025-02-22 08:43:49,865 - INFO -   entailment: 0.598
2025-02-22 08:43:49,865 - INFO -   contradiction: 0.201
2025-02-22 08:43:49,866 - INFO -   neutral: 0.201
2025-02-22 08:43:49,875 - INFO - 
Example 4:
2025-02-22 08:43:49,876 - INFO - Premise: Children are playing soccer in the par
k.
2025-02-22 08:43:49,876 - INFO - Hypothesis: Kids are engaged in outdoor sports.
2025-02-22 08:43:49,877 - INFO - Expected: entailment
2025-02-22 08:43:49,878 - INFO - Predicted: entailment
2025-02-22 08:43:49,879 - INFO - Probabilities:
2025-02-22 08:43:49,880 - INFO -   entailment: 0.709
2025-02-22 08:43:49,880 - INFO -   contradiction: 0.107
```

```
2025-02-22 08:43:49,881 - INFO -   neutral: 0.184
2025-02-22 08:43:49,891 - INFO -
Example 5:
2025-02-22 08:43:49,891 - INFO - Premise: The restaurant is packed with customer
s.
2025-02-22 08:43:49,892 - INFO - Hypothesis: The restaurant is closed today.
2025-02-22 08:43:49,892 - INFO - Expected: contradiction
2025-02-22 08:43:49,893 - INFO - Predicted: contradiction
2025-02-22 08:43:49,894 - INFO - Probabilities:
2025-02-22 08:43:49,894 - INFO -   entailment: 0.282
2025-02-22 08:43:49,895 - INFO -   contradiction: 0.386
2025-02-22 08:43:49,896 - INFO -   neutral: 0.332
2025-02-22 08:43:49,905 - INFO -
Example 6:
2025-02-22 08:43:49,906 - INFO - Premise: A student is writing notes in class.
2025-02-22 08:43:49,906 - INFO - Hypothesis: The student understands the materia
l.
2025-02-22 08:43:49,907 - INFO - Expected: neutral
2025-02-22 08:43:49,907 - INFO - Predicted: contradiction
2025-02-22 08:43:49,908 - INFO - Probabilities:
2025-02-22 08:43:49,909 - INFO -   entailment: 0.258
2025-02-22 08:43:49,910 - INFO -   contradiction: 0.399
2025-02-22 08:43:49,910 - INFO -   neutral: 0.343
2025-02-22 08:43:49,919 - INFO -
Example 7:
2025-02-22 08:43:49,920 - INFO - Premise: The chef is preparing pasta in the kitc
hen.
2025-02-22 08:43:49,920 - INFO - Hypothesis: Someone is cooking food.
2025-02-22 08:43:49,921 - INFO - Expected: entailment
2025-02-22 08:43:49,921 - INFO - Predicted: entailment
2025-02-22 08:43:49,922 - INFO - Probabilities:
2025-02-22 08:43:49,923 - INFO -   entailment: 0.868
2025-02-22 08:43:49,924 - INFO -   contradiction: 0.046
2025-02-22 08:43:49,924 - INFO -   neutral: 0.087
2025-02-22 08:43:49,933 - INFO -
Example 8:
2025-02-22 08:43:49,934 - INFO - Premise: The sky is clear and blue today.
2025-02-22 08:43:49,934 - INFO - Hypothesis: It is raining heavily.
2025-02-22 08:43:49,935 - INFO - Expected: contradiction
2025-02-22 08:43:49,936 - INFO - Predicted: entailment
2025-02-22 08:43:49,936 - INFO - Probabilities:
2025-02-22 08:43:49,937 - INFO -   entailment: 0.773
2025-02-22 08:43:49,938 - INFO -   contradiction: 0.106
2025-02-22 08:43:49,938 - INFO -   neutral: 0.121
2025-02-22 08:43:49,947 - INFO -
Example 9:
2025-02-22 08:43:49,947 - INFO - Premise: A person bought a new laptop.
2025-02-22 08:43:49,948 - INFO - Hypothesis: They got it from Amazon.
2025-02-22 08:43:49,949 - INFO - Expected: neutral
2025-02-22 08:43:49,949 - INFO - Predicted: contradiction
2025-02-22 08:43:49,950 - INFO - Probabilities:
2025-02-22 08:43:49,951 - INFO -   entailment: 0.291
2025-02-22 08:43:49,951 - INFO -   contradiction: 0.398
2025-02-22 08:43:49,952 - INFO -   neutral: 0.311
2025-02-22 08:43:49,961 - INFO -
Example 10:
2025-02-22 08:43:49,961 - INFO - Premise: The train arrived 30 minutes late.
2025-02-22 08:43:49,962 - INFO - Hypothesis: The train was delayed.
2025-02-22 08:43:49,963 - INFO - Expected: entailment
2025-02-22 08:43:49,963 - INFO - Predicted: entailment
```

```
2025-02-22 08:43:49,964 - INFO - Probabilities:
2025-02-22 08:43:49,965 - INFO -   entailment: 0.356
2025-02-22 08:43:49,965 - INFO -   contradiction: 0.350
2025-02-22 08:43:49,966 - INFO -   neutral: 0.294
2025-02-22 08:43:49,975 - INFO -
Example 11:
2025-02-22 08:43:49,975 - INFO - Premise: The museum is open on weekends.
2025-02-22 08:43:49,976 - INFO - Hypothesis: The museum is closed every day.
2025-02-22 08:43:49,976 - INFO - Expected: contradiction
2025-02-22 08:43:49,977 - INFO - Predicted: entailment
2025-02-22 08:43:49,977 - INFO - Probabilities:
2025-02-22 08:43:49,979 - INFO -   entailment: 0.375
2025-02-22 08:43:49,979 - INFO -   contradiction: 0.358
2025-02-22 08:43:49,980 - INFO -   neutral: 0.268
2025-02-22 08:43:49,989 - INFO -
Example 12:
2025-02-22 08:43:49,989 - INFO - Premise: A woman is walking her dog.
2025-02-22 08:43:49,990 - INFO - Hypothesis: The dog is brown in color.
2025-02-22 08:43:49,990 - INFO - Expected: neutral
2025-02-22 08:43:49,991 - INFO - Predicted: contradiction
2025-02-22 08:43:49,991 - INFO - Probabilities:
2025-02-22 08:43:49,992 - INFO -   entailment: 0.289
2025-02-22 08:43:49,993 - INFO -   contradiction: 0.410
2025-02-22 08:43:49,994 - INFO -   neutral: 0.300
2025-02-22 08:43:50,002 - INFO -
Example 13:
2025-02-22 08:43:50,003 - INFO - Premise: The movie theater is showing new releas
es.
2025-02-22 08:43:50,003 - INFO - Hypothesis: Films are being screened.
2025-02-22 08:43:50,004 - INFO - Expected: entailment
2025-02-22 08:43:50,005 - INFO - Predicted: entailment
2025-02-22 08:43:50,005 - INFO - Probabilities:
2025-02-22 08:43:50,006 - INFO -   entailment: 0.711
2025-02-22 08:43:50,007 - INFO -   contradiction: 0.132
2025-02-22 08:43:50,007 - INFO -   neutral: 0.157
```

Cool and Interesting Insights!

Confidence Patterns

1. High Confidence Successes

- Most successful entailment predictions had very high confidence (>0.7)
- Example: "chef preparing pasta" → "someone cooking food" (86.8% confidence)
- Example: "movie theater showing releases" → "films being screened" (71.1% confidence)

2. Challenging Cases

- Model struggles most with neutral relationships, often misclassifying them as contradictions
- Borderline cases show very close probability distributions across all three classes
- Example: "train delayed" case had almost equal probabilities (35.6%, 35.0%, 29.4%)

Pattern Analysis

1. Strong Performance:

- Direct logical entailments (cooking→food preparation)
- Simple activity relationships (playing soccer→outdoor sports)
- Direct contradictions with clear opposing statements

2. Common Mistakes:

- Over-predicting entailment for temporal relationships
- Struggling with neutral cases involving additional details
- Difficulty with implicit contradictions

Specific Weaknesses

1. Context Understanding:

- Failed on "clear sky" → "heavy rain" (predicted entailment with 77.3% confidence)
- Struggled with context-dependent relationships

2. Subtle Relationships:

- Poor performance on neutral cases requiring world knowledge
- Example: "student writing notes" → "understanding material"
- Example: "laptop purchase" → "Amazon purchase"

The model shows promising performance on straightforward relationships but needs improvement in handling subtle distinctions and world knowledge integration.

Thank You! 🤗