Task 1: Sentence Transformer Model Implementation

The objective of Task 1 was to encode input sentences into fixed-length embeddings using a pre-trained transformer encoder. The model chosen was 'all-MiniLM-L6-v2', a compact six-layer transformer producing 384-dimensional sentence embeddings. This architecture balances performance and speed, ideal for real-time or low-resource environments. The SentenceTransformer library was used to load the model and apply mean pooling over token embeddings, excluding padding. This pooling strategy produces one fixed-length embedding per sentence, capturing its semantic content and enabling its use in downstream classification, similarity comparison or clustering tasks

Task 2: Multi-Task Learning Expansion

In Task 2, the architecture was extended to support multi-task learning (MTL) by adding two independent task specific heads (linear output).

- Task A corresponds to Topic / Sentence Classification, and
- Task B corresponds to Sentiment Analysis.

A task flag routes each input through the appropriate head. Both heads are fed from a shared encoder (MiniLM) to maximize generalization and parameter efficiency. Mean-pooled sentence embeddings serve as input to both heads, ensuring uniformity across tasks.

Input Sentence -> Shared Transformer (MiniLM) ->Mean-pooled sentence embeddings
> Task A Head -> [Topic Logits]
> Task B Head -> [Sentiment Logits]

Task 3: Training Considerations

Scenario 1: Entire network frozen
This approach is useful for zero-shot or feature extraction as no model weights are updated. However, no fine-tuning occurs as the entire network is frozen.

Scenario 2: Only transformer backbone frozen
Common in limited data settings. The backbone's parameters remain fixed; only the task-specific heads are updated.

> This strategy leverages high-quality pre-trained language representations. In this case, just the head layers adapt to new tasks while preserving language knowledge. It reduces computational load during training and helps avoid overfitting when the available task-specific data is limited. However, if the tasks differ significantly from the pre-training

data or there is enough data , unfreezing some or all of the backbone might be beneficial.

Scenario 3: Freeze one head
Used in continual learning scenarios with an intent to preserve performance of one task while training the other(s). This approach helps to avoid interference between tasks during multi-task training. At the same time, frozen task might not fully benefit from joint training.

Transfer Learning Design Process

1. **Model Selection:**

In general, following parameters play a key role in selection of pre-trained models for transfer learning.

- Model (large vs. small) selection and embedding dimension
    - Large model captures richer semantics but risks over-fitting in a relatively small labeled dataset.
- Training objective of the base model
    - Ex. next-sentence prediction, contrastive learning or generative objectives (Sequence to Sequence, Causal Language Modeling)
- Domain of pre-trained model
    - General purpose or specialized such as biomedical, legal, financial.
- Inference speed and memory footprint
    - Deployability in edge vs cloud or on-premise environments.
- Community usage and support (including in-general feedback)
    - Generally high download models over a period of time are typically more robust and better documented.

For our MTL scenario with a very-small number of labeled data per task and running on constrained CPU infrastructure, I deliberately chose all-MiniLM-L6-v2. Its design offered a favorable trade off between semantic capability and resource requirements. I avoided heavyweight models due to over-fitting risks and memory req.

The encoder's fine-tuning strategy significantly impacts learning, especially in data-scarce scenario. I evaluated three freezing strategies:

1. Freeze All Layers Except Heads: A minimal learning setup. This variant showed poor generalization beyond the training set.

2. Unfreeze Only Last Layer + Heads: Allowed minor adaptation while preserving general semantic representations. However, performance varied across runs. The limited trainable surface proved insufficient to predict accurate results for topic classification and sentiment analysis over a new / unseen dataset. Hence I dropped this approach.

3. **Unfreeze Last Two Layers + Heads (Final)**: This approach yielded consistent and stable results. Layers 4 and 5 were unfrozen to allow domain adaptation, while the earlier layers remained frozen to preserve general language representations. With limited data, this approach provided an ideal balance of adaptation and regularization. I implemented this strategy in the code that I shared.

Task 4: Multi-Task Training Loop Implementation

Training within a MTL framework varies based on MTL strategy as it could be synchronous, asynchronous and curriculum MTL to name a few. Aligning with each of these, training strategy will change. In our context, we followed synchronous MTL.

Detail steps are:

- Data Loading: load sample from each task specific using DataLoaders (PyTorch). These loaders prepare batched data for Task A (Topic) and Task B (Sentiment)
- Forward Pass - Refer to sequence of computation that takes input text and converts it into task specific predictions.
- Tokenization → Encoder → Pooling (Mean Pool) → Task Head → Logits

  pa = model (sa, "A") #Topic logits

  pb = model(sb, "B") #Sentiment logits

- Loss computation: To compute task specific classification losses and later, I used nn.CrossEntropyLoss() to compute the difference between predicted logits and actual target labels.
- Backward pass (loss.backward()): Computes gradient of loss w.r.t. trainable parameters.

  It accumulates gradients through the model layer that we have.

- Parameter update (optimizer.step()): Updates the model's weight based on gradients. It also ensures gradient buffers are cleared after each step

  optimizer.zero_grad()

Metrics Tracking:

- We can track metric such as:
  - Accuracy
  - F1 Score
  - Precision and
  - Recall

Task 5: Evaluation and Insights

Evaluation was conducted on both seen and unseen data using the final trained model. The model was placed in evaluation mode using model.eval(), and predictions were made within a torch.no_grad() context to disable gradient tracking. Predictions were extracted using argmax from logits returned by the corresponding task head. Results were mapped back to human-readable classes using predefined dictionaries.

The configuration with the last two layers and both heads unfrozen demonstrated robust behavior, accurately predicting topic and sentiment across both seen and novel inputs. The one-layer-unfrozen variant yielded inconsistent generalization.

This evaluation confirmed the hypothesis that lightly fine-tuning high-capacity pretrained transformers offers the best tradeoff between performance and overfitting in few-shot learning scenarios.