ML Apprentice Take-Home Assessment

Sentence Transformers & Multi-Task Learning

1. Overview

This submission implements a sentence embedding model extended for multi-task learning (MTL) using PyTorch and Hugging Face Transformers.

The solution supports:

- Task A: Topic classification using the AG News dataset.

- Task B: Sentiment classification using the SST-2 dataset.

Backbone model: sentence-transformers/all-MiniLM-L6-v2

Includes a complete training loop, metrics, and prediction samples.

2. Task 1: Sentence Transformer Implementation

Architecture:

- Backbone: sentence-transformers/all-MiniLM-L6-v2

- Projection: 384 → 256 using Linear → Tanh

- Pooling: Mean pooling with attention mask

- Normalization: L2

Tokenizer: AutoTokenizer.from_pretrained(BACKBONE)

Embeddings are printed using sentence_embeddings()

Explanation:

The designed model processes inputs through a frozen MiniLM backbone and then applies masked mean-pooling to focus on actual tokens. To shrink the dimension of the vector to 256, the mean-pooled output is processed through a nn linear layer followed by a tanh activation. Finally, the resulting vector is L2-normalized to have a unit length, making it suitable for cosine similarity calculations. The final output is a 256-dimensional embedding that's optimized for fast inference and similarity searches.

3. Task 2: Multi-Task Learning Expansion

Model Structure:

- Shared encoder: SentenceEncoder

- Heads: Two task-specific heads using nn.ModuleDict

Dataset Handling:

- AG News: text field, 4 classes

- SST-2: sentence field, 2 classes

Label Definitions:

TOPIC_LABELS = {0: 'World', 1: 'Sports', 2: 'Business', 3: 'Sci/Tech'}

SENTIMENT_LABELS = {0: 'Negative', 1: 'Positive'}

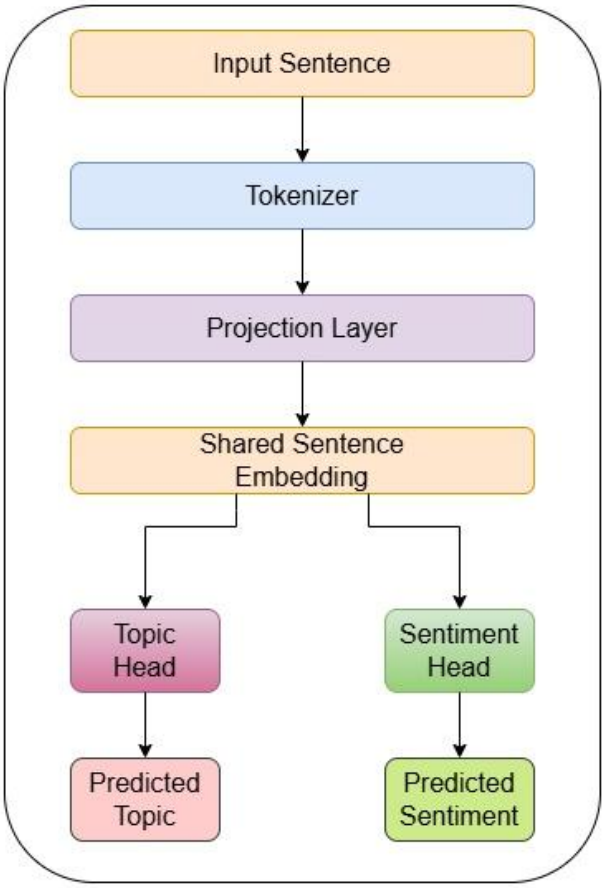
Explanation:

To expand the sentence-transformer into a multi-task learning model, no changes were made in the shared SentenceEncoder. It still performs masked mean pooling, a projection with tanh, and L2 normalization to produce a single 256-dimensional embedding for every sentence.

On top of this encoder, I added a wrapper - MultiTaskModel to perform classification and predict the sentiment using nn.ModuleDict() with 2 heads (using nn.Linear()).

The model's forward method takes an additional "task" argument: it encodes the sentence once with the shared backbone and then routes the resulting embedding through the appropriate head, returning task-specific logits which further will be used to predict the sentiment or topic.

Below mentioned is the overall architecture diagram:



4. Task 3: Training Considerations

Freezing Scenarios:

- Freeze entire network

- Freeze backbone only

- Freeze one task head

Transfer Learning:

- Model: all-MiniLM-L6-v2

- Unfreeze: last transformer layers + heads

Training Configuration:

- Optimizer: AdamW

- Learning Rate: 2e-5

- Scheduler: Linear

- Loss: CrossEntropyLoss

- Epochs: 3

Explanation:

Model Training:

- **Fully Frozen Network:** When the entire network will be frozen, the new model will retain only its pre-train knowledge as the parameters are not getting updated to the training data. Zero-shot inference use cases can be considered as an example for this type of training.
- **Transformer Backbone is Frozen:** As we are keeping the backbone frozen, we are going to train the remaining parts of the model, such as the head(s) as part of the network. By performing this, we are going to utilize the knowledge of backbone to perform a similar or new task. We can apply this approach when we have limited task-specific data. This specific approach is cheap to train and has less forgetting as long as the domain is not totally different.
- **One of the Task Specific Heads are Frozen:** Freezing either of the task heads i.e., the final prediction layers while fine-tuning a model means we are keeping those prediction layers fixed during the training process while the rest of the model continues to learn and adapt. This forces the model to learn new features using the previously frozen layers, but the learned representations are then used to make predictions based on those frozen heads. This approach is generally used when we want to preserve the performance of a task that has been trained well already. This approach is also helpful when we have a small or noisy dataset as it avoids overfitting.

Transfer Learning:

- **Pre-trained Model Choice:** Based on the tasks that I performed above, I would prefer FinBERT as the data is related to financial domain. Another alternate which is not specific to a domain is DeBERTa V3.
- **Layers to freeze or Unfreeze:** Freezing or unfreezing layers of a transformer depends on the task we are dealing with. If the new task and originally pre-trained transformer task are somewhat similar, it is better to freeze the first layers and fine-tune the last layers as the early layers of the transformer model captures general language patterns. Fine tuning the model on an unfrozen last layers allows the model to adapt its task specific understanding on the new data.
  If the new task is different from the pre-trained model task, it may be better to freeze the last layers of the backbone and fine-tune first layers.
- **Rationale:** Since my current model (MiniLM-L6-v2) is a lightweight transformer, for more better predictions, I would prefer FinBERT or DeBERTa because these are advanced and has more number of parameters which will the model in understanding the input in a better way compared to the lightweight model. Compared to FinBERT and MiniLM, DeBERTa uses enhanced masked encoding which is used to learn robust and meaningful representations of the data.

5. Task 4: Training Loop Implementation

Logic:

- Iterates over both tasks per epoch

Metrics:

- Accuracy, Precision, Recall, F1-score

6. Design Justification & Insights

Backbone: Lightweight and fast transformer

Mean pooling: Simplicity and stability

Shared encoder: Efficient for MTL

Split logic: Balanced splits

Unified loop: Simpler tracking and training

7. Setup Instructions

Environment Setup:

pip install -r requirements.txt

Docker:

docker build -t mtl-nlp .

Files:

- mtl_model_finetuning.ipynb: full solution

- requirements.txt

- Dockerfile

- README.md

- ML_Apprentice_Structured_Documentation.pdf