Fetch Assessment - ML Apprentice

Manan Abbott

May 20, 2025

1 Task 1: Sentence Transformer Implementation

1.1 High-level Design

I implemented a sentence-transformer model that converts input sentences into fixed-length embeddings by building on a pretrained Transformer backbone:

- **Pretrained backbone.** I used Hugging Face's `sentence-transformers/all-MiniLM-L6-v2`, which provides a compact 384-dimensional per token encoder.
- **CLS-token pooling.** After tokenization, I extract the `[CLS]` token embedding as the sentence representation.
- **Projection layer.** To control embedding dimension and add complexity to the model, I add

  $$\texttt{projection} = \texttt{Linear(hidden\_size, projection\_dim)}\,,$$

  mapping the backbone's hidden size (384) to a user-specified `projection_dim` (e.g. 256).
- **L2 normalization.** When normalization parameter is set to true, I calculate the Euclidean normalization so that all embeddings lie on the unit sphere, improving cosine-similarity comparisons.
- **Device handling & persistence.** The model class constructor automatically selects CUDA/MPS/CPU based on the device's availability. I have included `save` and `load` methods to checkpoint both the Transformer and projection weights.

1.2 Demonstration

The encoder's demonstration is in `notebooks/task1.ipynb` as follows:

```
from src.embedding_model import EmbeddingModel
model = EmbeddingModel()

sent1 = ["The temperature today is 71 degrees Fahrenheit."]

# Get the embedding for the sentence
embedding1 = model.encode(sent1)
```

```
print("Sentence:", sent1)
print("Embedding:", embedding1)
print("Embedding shape:", embedding1.shape) # [1,384]
```

1.3 Architectural Choices Beyond the Backbone

- **MiniLM backbone.** Chosen for a favorable trade-off between speed, memory footprint, and semantic accuracy.
- **Projection layer.** Allows the user to tailor embedding dimensionality to downstream tasks, balancing expressiveness and computational cost. For task 1, no projection layer was used to ensure reproducibility as the projection layers has to be trained.
- **Normalization.** Ensures embeddings are uniformly scaled, which improves numerical stability in downstream operations.

2 Task 2: Multi-Task Learning Expansion

2.1 Problem Description

In Task 2, I extended the pretrained sentence-transformer encoder with two distinct classification heads, so that a single model can solve two NLP tasks simultaneously:

- **Task A head (4-way news classification).** For an input sentence, this head predicts one of four arbitrary news categories.
- **Task B head (3-way sentiment analysis).** For the same input sentence, this head predicts its sentiment as positive, neutral, or negative.

2.2 Model Architecture

I built a `MultiTaskModel` class that wraps the pretrained sentence-transformer (`EmbeddingModel`) and attaches two task-specific classification heads to the embedding model:

- **Shared backbone:** The backbone of the model is created using our embedding model from Task 1. In case if there is a cached version available, load it.

  ```
  self.backbone = EmbeddingModel(projection_dim=256, normalize=True)
  if backbone_pth:
      self.backbone.load(backbone_pth)
  ```

  This backbone provides us with the `[CLS]` token (256 dimensional vector) for each input sentence.

- **Embedding dimension.** We determine the output size of the of our backbone based on if there is a projection layer or not.

$$\texttt{embed\_dim} = \begin{cases} \texttt{self.backbone.projection.out\_features}, & \text{if projection exists} \\ \texttt{self.backbone.model.config.hidden\_size}, & \text{otherwise} \end{cases}$$

  Essentially so both heads consume the correct input size.

- **Task A head (4-way news classification).**

$$\text{embed\_dim} \xrightarrow{\text{Linear(256}\to\text{128)}} 128 \xrightarrow{\text{ReLU}} 128 \xrightarrow{\text{Dropout(0.2)}} 128, \xrightarrow{\text{Linear(128}\to\text{32)}} 32 \xrightarrow{\text{ReLU}} 32 \xrightarrow{\text{Dropout(0.2)}} 32 \xrightarrow{\text{Linear(32}\to\text{4)}} 4$$

- **Task B head (3-way sentiment analysis).**

$$\text{embed\_dim} \xrightarrow{\text{Linear(256}\to\text{128)}} 128 \xrightarrow{\text{ReLU}} 128 \xrightarrow{\text{Dropout(0.2)}} 128, \xrightarrow{\text{Linear(128}\to\text{32)}} 32 \xrightarrow{\text{ReLU}} 32 \xrightarrow{\text{Dropout(0.2)}} 32 \xrightarrow{\text{Linear(32}\to\text{3)}} 3$$

2.3 Forward Pass & Prediction

- `forward(sentences):`
  1. Encode with the shared backbone:

     `embeddings = self.backbone.encode(sentences)`

  2. Compute logits for each task:

$$(\texttt{logitsA}, \texttt{logitsB}) = (\texttt{self.taskA\_head(embeddings)}, \texttt{self.taskB\_head(embeddings)})$$

  3. Return both sets of logits.

- `predict(sentences):` Runs `forward` in `eval()` mode, applies softmax to get probabilities of every class and `argmax` to yield predictions with highest probability in each class.

2.4 Training Loop Helpers

The methods to assist multi-task training and evaluation are as follows:

- `train_epoch:` Iterates over batches of `(sentences, labelsA, labelsB)`, computes `loss = lossA(logitsA, labelsA) + lossB(logitsB, labelsB)`, backpropagates, and returns average loss.

- `eval_epoch:` In `no_grad` mode, evaluates the performance over the validation set and returns (avg_loss, accA, accB).

- `fit:` Performs epochs of training and validation, applies an optional scheduler, and checkpoints the best model based on validation loss.

2.5 Fine-Tuning Helpers

To support transfer-learning , there are freeze/unfreeze methods for each component of the model:

- `freeze_backbone()` / `unfreeze_backbone()`: Freeze/Unfreeze the backbone encoder.
- `freeze_projection()` / `unfreeze_projection()`: Freeze/Unfreeze the projection layer of the model, if there is any.
- `freeze_taskA_head()` / `unfreeze_taskA_head()`: Freeze/Unfreeze the news classification head.
- `freeze_taskB_head()` / `unfreeze_taskB_head()`: Freeze/Unfreeze the sentiment analysis (classificatoin) head.

These enable fine-tuning of the Transformer body, projection layer, or individual heads as needed during training.

3 Task 3: Training Considerations

3.1 Freezing Strategies

3.1.1 Freezing All Parameters

- Freezing all parameters means that neither the pretrained Transformer backbone nor the newly added classification heads receive any gradient updates.
- The model is used purely in a zero-shot setting where primary advantage of this approach is that the pretrained linguistic and semantic knowledge is preserved without risk of degradation, and it requires no training time or resources.
- The main purpose of this method is to establish a performance baseline: by measuring how random, untrained heads perform on top of fixed embeddings, we can quantify the value added by fine-tuning steps. It works as a sanity check.
- No training is performed here as there are not any gradient updates taking place. This is purely a zero-shot baseline.

3.1.2 Freezing the Transformer Backbone

- Freezing the Transformer backbone means that all pretrained Transformer layers remain fixed, while only the projection layer (if any) and both task heads receive gradient updates.
- This strategy preserves a robust, general-purpose sentence representations and prevents overfitting of the large backbone parameters when labeled data points are scarce.
- It speeds up training and reduces memory consumption since backpropagation does not flow through the Transformer layers, making it an effective first stage of fine-tuning, also called as ’warm-up’.
- Use an optimizer (e.g. Adam) on only the projection (if any) and both heads with a relatively high learning-rate (e.g. 1e−3), for 3–5 epochs or until validation loss plateaus.

3.1.3 Freezing a Single Task Head

- Freezing a single task head means that one classification head (e.g. Task B) remains fixed while the backbone, projection, and the other head continue to be trained.
- This approach prevents a converged or high-priority task head from dominating the shared gradient signal, freeing up capacity for the underperforming task.
- It enables flexible training strategies such as alternating freeze schedules—to balance performance across tasks.
- Train the backbone, projection, and unfrozen head. Use optimizer (e.g. Adam)with a smaller LR for the backbone (1e−5) and a higher LR for the head (1e−4)—for 3–5 epochs, optionally alternating which head is frozen every few epochs.

3.2 Proposed Transfer-Learning Strategy

1. **Pretrained model choice.** The `all-MiniLM-L6-v2` model was chosen because it offers a strong trade-off between parameter count, inference speed, and semantic understanding on general NLP tasks.
2. **Initial freeze plan.**
   - *Epochs 1–10:* Freeze the backbone completely; train only the projection layer and both heads with a moderate learning rate (e.g. 1e−3).
   - This stabilizes the heads on top of fixed, well-calibrated embeddings.
3. **Unfreezing schedule.**
   - *Epochs 11–20:* Unfreeze the top Transformer block (last 2 layers) alongside projection and heads; reduce backbone learning rate to 1e−5 while keeping learning rate of heads at 1e−4.
   - *Epochs 21-25:* Unfreeze all the parameters, lowering its learning rate for backbone to 1e−6 and heads & projection to 1e−5, to allow deeper adaptation without forgetting.
4. **Rationale.**
   - Freezing the backbone initially prevents the model from "forgetting" general language representations while allowing the new layers (projection + both heads) to warm up and learn task-specific mappings on top of those stable embeddings from the backbone before any deeper fine-tuning.
   - Layer-wise unfreezing with decreasing learning rates lets me refine specific semantic features for my tasks, while preserving robustness in lower layers.
   - Different learning rates across backbone and heads ensure that newly added layers adapt quickly to the tasks, while the backbone updates conservatively. This approach typically improves training stability and leads to better convergence.

3.3 Task 4: Training Loop

NOTE: I use a sample dataset of 120 samples created using generative AI to showcase the validity of my approach.

We have two assumptions about the data:

- The sentences data (news headlines) has two labels associated with it: first is the kind of news(1 out of 4 arbitrary classes) and second is its sentiment (positve, negative or neutral).
- `train_dataloader` gives us batches of training data (format: `(sentences, labelA, labelB)`) and `val_dataloader` gives us batches of validation data (format: `(sentences, labelA, labelB)`)

3.4 Design Assumptions

- **Equal loss weighting:** The two task losses are added together ($\ell = \ell_A + \ell_B$) to keep the training simple. If one task dominates, then we could introduce a weighting factor α: $\ell = \alpha\,\ell_A + (1-\alpha)\,\ell_B$.
- **Separate metrics:** I track loss and accuracy independently for Task A and Task B to monitor convergence and detect task imbalance.
- **Scheduler usage:** A learning-rate scheduler is stepped after each epoch, which helps smooth training and avoid rapid plateaus, especially when unfreezing backbone layers.
- **Checkpointing:** The model is saved whenever the combined validation loss $\ell_A + \ell_B$ improves, ensuring the best multi-task weights (model) is retained.