# **ML Apprentice Take-Home Assessment: Written Explanation -** [**Abhishek Pathak**](mailto:enkode@umich.edu)

## **Task 1: Sentence Transformer Implementation**

### **Objective**

The goal was to implement a **sentence transformer** capable of encoding sentences into **fixed-length embeddings**.

### **Framework Choice**

* **PyTorch** and **Huggingface Transformers** were chosen because they are the industry standards for building and fine-tuning transformer models.

### **Architecture Design**

* **Transformer Backbone**: Used a pre-trained bert-base-uncased model.
* **Embedding Strategy**: Applied **Mean Pooling** over the output token embeddings to generate the final sentence embedding.

### **Design Choices and Justifications**

* **Mean Pooling vs [CLS] token**:
  + Chose **Mean Pooling** because it tends to produce more stable and representative embeddings across various tasks, compared to using only the [CLS] token.
* **Tokenizer Usage**:
  + Used the tokenizer associated with the selected transformer backbone to maintain consistency between pre-training and downstream tasks.

### **Testing**

* Tested the implementation with a few sample sentences and confirmed that the model outputs consistent, fixed-size embeddings.

## **Task 2: Multi-Task Learning Expansion**

### **Objective**

Extend the sentence transformer model to support **Multi-Task Learning (MTL)**.

### **Tasks Defined**

* **Task A**: Sentence Classification (3 classes: e.g., "business", "sports", "technology").
* **Task B**: Sentiment Analysis (2 classes: "positive" or "negative").

### **Architecture Changes**

* **Shared Backbone**: The sentence embeddings are generated from the same transformer model.
* **Task-specific Heads**:
  + **Task A Head**: A two-layer MLP classifier for 3-class classification.
  + **Task B Head**: A two-layer MLP classifier for binary sentiment classification.

### **Design Decisions and Justifications**

* **Separate Heads**:
  + Different heads allow each task to specialize in learning its own output space without interfering with others.
* **Shared Embedding Layer**:
  + Sharing the transformer backbone leverages common linguistic understanding while keeping computational costs low.

## 

## 

## 

## 

## 

## **Task 3: Training Considerations**

### **Freezing Strategies and Implications**

1. **Freeze Entire Network**
   * Only train the task-specific heads.
   * Useful when labeled data is extremely limited.
   * Risk: The model might underfit if the backbone representations do not align well with the task.
2. **Freeze Only Transformer Backbone**
   * Fine-tune only the task-specific heads.
   * Useful when tasks are moderately similar to the pre-trained domain (e.g., general English text).
   * Benefits: Faster training and lower risk of catastrophic forgetting.
3. **Freeze Only One Task Head**
   * Freeze the head of a task where performance is already good.
   * Fine-tune only the underperforming task’s head.
   * Useful when trying to improve a particular task without hurting another task's performance.

### **Transfer Learning Scenario**

#### **Example Scenario**

* Adapting to the **medical domain** (e.g., clinical notes or research papers).

#### **Approach**

* **Pre-trained Model**: Use biobert-base-cased (a version of BERT pre-trained on biomedical data).
* **Layer Freezing Strategy**:
  + Freeze **lower layers** (which learn general grammar and syntax).
  + Fine-tune **higher layers** and **task-specific heads** (which learn domain-specific semantics).
* **Rationale**:
  + Maintain the fundamental language knowledge from pre-training.
  + Allow adaptation to domain-specific terminology without forgetting general language patterns.

## 

## **Task 4: Training Loop Implementation**

### **Assumptions Made**

* **Data**: Hypothetical, dummy sentences and labels were used for simulation.
* **Separate Losses**: Each task has its own loss function (CrossEntropyLoss).
* **Metrics**: Used simple accuracy as a metric for evaluation.

### **Forward Pass and Optimization**

* The training loop:
  + Selects the correct head based on the task.
  + Computes the loss.
  + Backpropagates the error.
  + Updates the model parameters accordingly.

### **Special Notes**

* Separate evaluation for each task was performed to monitor performance independently.
* The model training loop was kept simple and modular to handle extensions like different optimizers per task or weighted multi-task losses.

## **Environment Setup**

The project uses the following dependencies:

nginx

CopyEdit

torch

transformers

They are listed in requirements.txt for easy installation.

## 

## 

## **Docker Containerization**

A Dockerfile was created to containerize the environment for reproducibility:

* Python 3.10 slim base image.
* Installed all required libraries.
* Set up the default command to run train.py.

This ensures the project can be easily deployed or tested without manual setup.

# **Conclusion**

Through this project, modular and scalable code was designed to satisfy:

* Sentence-level embedding extraction.
* Multi-task learning with separate heads.
* Practical training strategies for transfer learning and freezing models.
* Clear, structured, and efficient implementation aligning with industry best practices.

Each design decision has been carefully made to balance flexibility, performance, and simplicity, while focusing on clarity as per the evaluation guidelines.