Assignment 3

Module Code: DAM202

TRANSFORMER ENCODER

Course Context

Prerequisite Knowledge: Transformer architecture theory, self-attention mechanism, multi-head attention, layer normalization, and positional encoding.

Learning Objectives:

- Implement and understand the components of a Transformer encoder from scratch or customize pre-trained encoders
- Apply encoder models to real-world NLP tasks
- Evaluate model performance using appropriate metrics
- Analyze learned representations and attention patterns

Assignment Overview

Task: Develop a **Transformer Encoder-based System** from scratch or by fine-tuning a pre-trained encoder model (BERT, RoBERTa, DistilBERT, …) on a domain-specific task of your choice.

Submission Format: A comprehensive project report with code, visualizations, and analysis.

Assignment Deadline: 22 November, 2025.

Detailed Requirements

Part A: Data Preparation and Exploration

1. **Dataset Selection and Justification**
    - Select a text classification dataset (e.g., sentiment analysis, topic classification, or domain-specific categorization)
    - Options include: IMDB movie reviews, Stanford Sentiment Treebank (SST-5), AG News, or a custom domain-specific corpus.
    - Provide statistical analysis: class distribution, text length distribution, vocabulary size, and dataset characteristics.
    - Perform train-test-validation split (60%-20%-20% or any suitable ratio) with stratification
    - Create an Exploratory Data Analysis (EDA) report with visualizations
2. **Tokenization and Vocabulary Analysis**
    - Implement tokenization using a chosen tokenizer (WordPiece, SentencePiece, or Byte-Pair Encoding)
    - Analyze token statistics: average sequence length, maximum sequence length, token frequency distribution
    - Document handling of out-of-vocabulary words and padding strategies
    - Generate a sample vocabulary report with token distributions

Part B: Model Architecture and Implementation

3. Encoder Architecture Design and Implementation

Option A: Build Encoder from Scratch (Recommended for comprehensive understanding)

Implement the following components:

- Multi-head self-attention mechanism with scaled dot-product attention
- Position-wise feed-forward networks (FFN) with ReLU/GeLU activation
- Positional encoding (sinusoidal or learned embeddings)
- Layer normalization and residual connections
- Complete encoder layer combining all sub-modules
- Stack of N encoder layers (recommend N=6 for balance between complexity and training efficiency)
- Output head: Use appropriate Layer with activation function (say Softmax for probability-based prediction) for final output. For computational effectiveness beam or greedy search are desired.

Option B: Fine-tune Pre-trained Encoder (Practical alternative)

- Load a pre-trained encoder model (BERT-base, DistilBERT, or RoBERTa)
- Freeze encoder weights initially, train only the classification head
- Implement layer-wise learning rate decay for fine-tuning

- Document architecture modifications and justifications

4. **Model Configuration Documentation**

    - Hyperparameter specifications: embedding dimension (d_model), number of heads, hidden layer size (d_ff), dropout rates
    - Training configuration: optimizer (Adam with learning rate scheduling), batch size, number of epochs, loss function
    - Regularization techniques: dropout, label smoothing, early stopping criteria
    - Hardware specifications and computational requirements

Part C: Training and Evaluation

5. **Model Training Protocol**

    - Implement complete training pipeline with:
        - Mixed precision training for efficiency (FP16)
        - Gradient accumulation for larger effective batch sizes
        - Learning rate warmup and decay scheduling
        - Checkpoint saving strategy (save best model based on validation accuracy)
    - Generate training logs: loss curves, accuracy curves, learning rate schedule visualization
    - Monitor and document: training time, convergence behavior, memory usage

6. **Comprehensive Model Evaluation**

- Test set performance metrics: accuracy, precision, recall, F1-score, macro-averaged F1
- Per-class performance analysis and confusion matrix visualization
- Comparison with baseline models (TF-IDF + SVM, LSTM baseline) [Optional but best for learning purpose]
- Generate performance report with statistical significance testing (if applicable)

7. **Attention Visualization and Interpretability Analysis**
    - Extract and visualize attention weights from multiple encoder layers
    - Create attention head visualizations showing which input tokens the model focuses on
    - Provide interpretability analysis: which words receive highest attention weights
    - Generate heatmaps for at least 5 sample predictions (If your model is for predicting)
    - Analyze failure cases: instances where model performs poorly and attention patterns

Part D: Advanced Analysis and Reporting

8. **Transfer Learning and Domain Adaptation Analysis**
    - If time permits: demonstrate transfer learning by fine-tuning on a second related task with less data
    - Document performance improvement due to pre-training
    - Analyze what linguistic knowledge the encoder learns

9. **Ablation Study**
    - Study impact of key architectural components:
        - Number of attention heads (experiment with 4, 8, 16)
        - Embedding dimension variations
        - Number of encoder layers
    - Present results in a comparative table with reasoning
10. **Comprehensive Final Report**
    - Executive summary (1-2 pages)
    - Literature review on encoder-based models
    - Detailed methodology section
    - Results with visualizations (plots, attention heatmaps, confusion matrices)
    - Interpretation of findings and attention analysis
    - Ablation study results
    - Limitations and future work
    - Code appendix and reproducibility notes

Deliverables

1. **Source Code**
    - Modular, well-documented Python code (with type hints)

- Separate files for: data processing, model architecture, training loop, evaluation, visualization
- Configuration file for hyperparameters (YAML or JSON)
- Requirements.txt file with all dependencies

2. **Documentation**
    - README with setup instructions and usage guide
    - Inline code comments explaining key concepts
    - Jupyter notebook demonstrating model usage on new samples
3. **Results and Visualizations**
    - Training curves (loss, accuracy, learning rate)
    - Attention weight visualizations (at least 10 examples)
    - Confusion matrix
    - Comparative performance table
4. **Written Report**
    - Follow academic paper format
    - Include all sections mentioned in Part D

Note:

This is just a reference for you to follow when confused. You are permitted to follow an appropriate assignment guide of your choice other than this one, but you must use the given topic only.