University of Central Missouri

Department of Computer Science & Cybersecurity

CS5710 Machine Learning

Fall 2025

Home Assignment 5

Student name: AJAY MUPPA

Submission Requirements:

- Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
- Comment your code appropriately ***IMPORTANT.***
- Any submission after provided deadline is considered as a late submission.

Part A — Short-Answer

1. Positional Encoding Concepts

a) Why do we need positional encodings in transformer models?

Transformers are built from self-attention, which treats input tokens as a set — attention by itself is permutation-invariant. Positional encodings inject information about token order/relative position so the model can distinguish sequences (e.g., "dog bites man" vs "man bites dog"). Without positional encodings the model loses temporal/ordering structure critical for language, time series, and many sequence tasks.

b) Describe two key requirements for a good positional encoding scheme.

- **Distinctiveness across positions** — encodings for different positions should be distinguishable so the model can learn position-specific behaviors.
- **Generalization to unseen lengths / helpful relative information** — either provide a mechanism to extrapolate to longer sequences (e.g., sinusoidal encodings have predictable behavior beyond training length) or encode relative distances (relative positional embeddings) so the model can infer relationships between positions even if absolute position differs.

c) What does it mean for the positional encoding matrix M to be *unitary* and *norm-preserving*?

- **Unitary**: a matrix P is unitary (orthogonal in real-valued case) if $P^{\intercal}P = I$. That means the transform preserves inner products and angles between vectors.
- **Norm-preserving**: applying the encoding does not change the magnitude (norm) of the embedded vectors. Norm preservation avoids artificially scaling activations which could destabilize training or change attention magnitudes. A unitary positional transform both preserves norms and preserves relative geometry (angles, dot products).

2. Attention Mechanism

a) Define "attention score" and explain how it determines the weight of each token.

An *attention score* is an alignment scalar that measures compatibility between a query (usually derived from the current token or decoder state) and a key (representing another token). Commonly computed as $\mathrm{dot}(Q, K)$ (sometimes scaled by $\sqrt{d_k}$). These scores are converted into normalized weights (via softmax) so that tokens with higher compatibility receive higher attention weight (i.e., influence the output more).

b) What mathematical operation is applied to convert alignment scores into attention weights?

Apply the **softmax** function across the scores for a given query to convert raw alignment scores into a probability distribution (nonnegative weights summing to 1).

c) How is the *context vector* computed from these weights and values?

The **context vector** (or attention output) is the weighted sum of the value vectors V where weights are the softmax-normalized attention weights. Mathematically: Context $= \mathrm{softmax}(QK^{\intercal}/\sqrt{d_k})V$.

3. Multi-Head Attention

a) What is the main advantage of using multiple attention heads?

Multiple heads let the model attend to different representation subspaces and different positions simultaneously (e.g., one head learns syntactic relations, another learns long-range semantic ties). This enriches representational capacity compared to a single attention map.

b) How does splitting Q, K, and V across different subspaces improve model representation?

By projecting Q/K/V into smaller subspaces (via separate learned linear maps) each head focuses on distinct features and patterns. This enables specialized attention: different heads can focus on different distance ranges, token types, or syntactic functions, then their outputs are combined to form a richer representation.

c) After multi-head attention, why is concatenation followed by another linear projection necessary?

Concatenation aggregates the information from all heads into one vector. A final linear projection (learned) then mixes and recombines features from different heads, enabling the model to form the final representation that leverages complementary information from heads.

4. Ethical Foundations

a) Explain why *ethics* is not the same as *laws* or *feelings*.

- Laws are codified rules enforced by institutions; ethics is a broader, reflective discipline about right/wrong that can influence law but also go beyond it. Something can be legal yet unethical (or illegal but ethically justified in certain views).
- Feelings are personal, subjective responses; ethics uses reasoned frameworks (principles, consistent rules, consequences) to evaluate actions beyond immediate emotion. Thus ethics demands justification and argumentation, not only sentiment.

b) Briefly describe two classical ethical theories (e.g., utilitarianism and deontology) and how they would handle an AI decision scenario.

(Scenario: an AI hiring filter must choose candidates with limited data.)

- **Utilitarianism (consequentialism)**: choose action that maximizes aggregate wellbeing — pick the policy that on average yields the best outcomes (e.g., highest expected fairness & productivity). Could justify demographic-blind scoring if it optimizes outcomes, but must weigh societal harms.
- **Deontology (duty/rules-based)**: judge actions by adherence to duties/rules (e.g., non-discrimination) rather than only outcomes. Would forbid knowingly using protected attributes or proxies even if predictive power helps.

c) Why do philosophers argue that no single ethical theory clearly "wins" in all contexts?

Theories prioritize different moral goods (outcomes, rights, duties, virtues). Real-world dilemmas involve trade-offs (efficiency vs fairness vs rights) and context-specific values, making one-size-fits-all approach insufficient.

5. Types of AI Harms

a) Define allocational harm and representational harm in AI systems.

- **Allocational harm**: unequal distribution of resources/opportunity by automated decisions (e.g., hiring filters excluding qualified minorities).

- **Representational harm**: stereotyping, exclusion, or offensive depictions that harm dignity and cultural representation (e.g., translation systems masculinizing words, facial-recognition failing on certain skin tones).

b) Provide an example of each from real-world applications (e.g., translation, hiring, or facial recognition).

- Allocational: automated resume screening systematically downgrading applicants from a disadvantaged group leading to fewer interview offers.
- Representational: language models that translate gender-neutral terms into a gendered language using incorrect stereotypes.

c) Why is representational harm often harder to measure than allocational harm?

Allocational harms map to measurable outcomes (hiring rates, loan approvals). Representational harms involve identity, dignity, and cultural nuance — harder to quantify, context-dependent, and often assessed qualitatively.

6. Sources of Dataset Bias

a) List three reasons why bias arises during data collection or annotation in AI datasets.

- **Sampling bias** — dataset not representative of the target population (e.g., web text skewed to English/America).
- **Labeler bias** — annotator cultural or subjective bias influences labels.
- **Measurement/collection process** — sensors, UI, or scraping methods systematically exclude or misrepresent groups.

b) What kinds of data or groups tend to be under-represented in large language datasets?

Minority languages/low-resource dialects, marginalized social groups, or non-Western cultural content are often under-represented.

c) How can bias amplification occur even after initial data preprocessing?

If imbalanced classes are oversampled or model objective magnifies common patterns (e.g., cross-entropy leading model to over-rely on majority patterns), initial biases can be exaggerated — model may more strongly prefer the majority signal than the raw data suggested.

7. Safety, Security, and Privacy

a) Define data poisoning and describe how it can manipulate a model's predictions.

Data poisoning is the insertion of maliciously crafted training examples so that the trained model behaves incorrectly for specific inputs (e.g., causing a classifier to misclassify a target sample or to leak secrets). It manipulates model decision boundaries to change predictions at test time.

b) What are the ethical implications of model memorization (e.g., GPT-2 reproducing private or copyrighted text)?

If a model memorizes and reproduces private or copyrighted text (e.g., GPT-2-level verbatim reproduction), it violates privacy and IP rights. Ethical issues include leaking PII, exposing proprietary text, and causing legal liability.

c) How does model stealing threaten privacy and intellectual property in AI research?

Model stealing replicates a model's functionality (via query access), risking intellectual property loss and enabling attackers to reproduce a model without training costs — undermining the original owners' investment and potentially enabling misuse of the stolen model.

Part B — Coding

Q1. Compute Scaled Dot-Product Attention (Python)

Write a Python function to compute the scaled dot-product attention given query Q, key K, and value Vmatrices.

- Use NumPy for matrix operations.
- Normalize scores using softmax.
- Return both attention weights and the resulting context vector.

Hint:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Code:

In [1]:

```python
# Q1_scaled_attention.py
import numpy as np

def softmax(x, axis=-1):
    # numerically stable softmax
    x_max = np.max(x, axis=axis, keepdims=True)
    e = np.exp(x - x_max)
    return e / np.sum(e, axis=axis, keepdims=True)

def scaled_dot_product_attention(Q, K, V, mask=None):
    """
    Compute scaled dot-product attention.
    Inputs:
      Q: (batch, seq_q, d_k)
      K: (batch, seq_k, d_k)
      V: (batch, seq_k, d_v)
      mask: optional (batch, seq_q, seq_k) with 0 for allowed positions and -inf (or large negative) for masked
    Returns:
      attention_weights: (batch, seq_q, seq_k)
      context: (batch, seq_q, d_v)
    """
    d_k = Q.shape[-1]
    # raw scores: (batch, seq_q, seq_k)
    scores = np.matmul(Q, np.swapaxes(K, -1, -2)) / np.sqrt(d_k)
    if mask is not None:
        # assume mask contains True for positions to mask, or additive mask
        scores = np.where(mask, -1e9, scores)
    # attention weights
    attn_weights = softmax(scores, axis=-1)
    # context: weighted sum over values
    context = np.matmul(attn_weights, V)  # (batch, seq_q, d_v)
    return attn_weights, context
```

```python
# small test
if __name__ == "__main__":
    np.random.seed(1)
    B = 2
    seq_q = 3
    seq_k = 4
    d_k = 8
    d_v = 6
    Q = np.random.randn(B, seq_q, d_k)
    K = np.random.randn(B, seq_k, d_k)
    V = np.random.randn(B, seq_k, d_v)
    attn_w, ctx = scaled_dot_product_attention(Q, K, V)
    print("attn_w shape:", attn_w.shape)  # (2,3,4)
    print("ctx shape:", ctx.shape)        # (2,3,6)
```

Output:

```
attn_w shape: (2, 3, 4)
ctx shape: (2, 3, 6)
```

Code Link:

https://colab.research.google.com/drive/1Xh4etPEe4R7S1ODahFcKZ-6Gk66oG_5I?usp=sharing

Github Link:

https://github.com/ajaymuppa/ML_Homework_5/tree/main

Q2. Implement Simple Transformer Encoder Block (PyTorch)
Implement a simplified transformer encoder block in PyTorch with the following components:

- Multi-head self-attention layer
- Feed-forward network (2 linear layers with ReLU)
- Add & Norm layers

Sub-tasks:
a) Initialize dimensions $d_{model} = 128, h = 8$.
b) Add residual connections and layer normalization.
c) Verify the output shape for a batch of 32 sentences, each with 10 tokens.

Code:

In [2]:

```python
# Q2_transformer_encoder.py
import torch
import torch.nn as nn
import torch.nn.functional as F

class SimpleMultiHeadSelfAttention(nn.Module):
    def __init__(self, d_model, num_heads):
        super().__init__()
        assert d_model % num_heads == 0, "d_model must be divisible by num_heads"
        self.d_model = d_model
        self.num_heads = num_heads
        self.d_head = d_model // num_heads

        # linear projections for Q, K, V and output
        self.W_q = nn.Linear(d_model, d_model)
        self.W_k = nn.Linear(d_model, d_model)
        self.W_v = nn.Linear(d_model, d_model)
        self.W_o = nn.Linear(d_model, d_model)

    def forward(self, x, mask=None):
        # x: (batch, seq_len, d_model)
        B, T, _ = x.size()
        Q = self.W_q(x)  # (B, T, d_model)
        K = self.W_k(x)
        V = self.W_v(x)

        # reshape to heads: (B, num_heads, T, d_head)
        def split_heads(tensor):
            return tensor.view(B, T, self.num_heads, self.d_head).transpose(1, 2)
        Qh = split_heads(Q)
        Kh = split_heads(K)
        Vh = split_heads(V)

        # scaled dot-product per head
        scores = torch.matmul(Qh, Kh.transpose(-2, -1)) / (self.d_head ** 0.5)  # (B, h, T, T)
        if mask is not None:
            # mask shape should be broadcastable to (B, 1, T, T), use True for masked positions
            scores = scores.masked_fill(mask.unsqueeze(1).bool(), float('-inf'))
        attn = F.softmax(scores, dim=-1)  # (B, h, T, T)
        out_heads = torch.matmul(attn, Vh)  # (B, h, T, d_head)
```

```
        # concat heads -> (B, T, d_model)
        out = out_heads.transpose(1, 2).contiguous().view(B, T, self.d_model)
        out = self.W_o(out)
        return out, attn  # return attention for debugging if desired

class TransformerEncoderBlock(nn.Module):
    def __init__(self, d_model=128, num_heads=8, d_ff=512, dropout=0.1):
        super().__init__()
        self.self_attn = SimpleMultiHeadSelfAttention(d_model, num_heads)
        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.ffn = nn.Sequential(
            nn.Linear(d_model, d_ff),
            nn.ReLU(),
            nn.Linear(d_ff, d_model)
        )
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, mask=None):
        # x: (B, T, d_model)
        attn_out, attn = self.self_attn(x, mask=mask)
        x = x + self.dropout(attn_out)    # residual + dropout
        x = self.norm1(x)

        ffn_out = self.ffn(x)
        x = x + self.dropout(ffn_out)     # residual + dropout
        x = self.norm2(x)

        return x, attn
```

```
# test / verification
if __name__ == "__main__":
    B = 32
    T = 10
    d_model = 128
    num_heads = 8
    x = torch.randn(B, T, d_model)

    block = TransformerEncoderBlock(d_model=d_model, num_heads=num_heads, d_ff=512)
    out, attn = block(x)  # out shape should be (32, 10, 128)
    print("out.shape:", out.shape)   # expect (32, 10, 128)
    print("attn.shape:", attn.shape) # expect (32, num_heads, 10, 10)
```

Output:

```
out.shape: torch.Size([32, 10, 128])
attn.shape: torch.Size([32, 8, 10, 10])
```