# University of Central Missouri

# Department of Computer Science & Cybersecurity

## CS5760 Natural Language Processing

## Fall 2025

## Homework 4.

**Student name: Ajith.Bonda**

## 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.
- Submit your GitHub link on the Bright Space.
- Comment your code appropriately **IMPORTANT.**
- Any submission after provided deadline is considered as a late submission.

GIT Link: https://github.com/Ajith0028/NLP_Assignment_4

**Part I. Short Answer**

**1) RNN Families & Use-Cases (Many-to-X)**
a) Map each task to the most suitable RNN I/O pattern and explain why (1–2 lines each):
• next-word prediction

• sentiment of a sentence

• NER

• machine translation
(Choose from: one-to-many, many-to-one, many-to-many aligned, many-to-many unaligned.)

Ans:

**Next-word prediction:** many-to-many (aligned) - each input token produces a next-token prediction aligned in time.

**Sentiment of a sentence:** many-to-one - full sequence processed to output a single sentiment label.

**NER:** many-to-many (aligned) - every input word requires a corresponding output tag.

**Machine translation:** many-to-many (unaligned) - input and output sequences differ in length, requiring an encoder–decoder setup.

b) In one sentence, explain how "unrolling" over time enables BPTT and weight sharing.

Ans: Unrolling converts the recurrent loop into a time-step chain, enabling BPTT and allowing the same weights to be reused at every step.

c) Give one advantage and one limitation of weight sharing across time in RNNs.

Ans: **Advantage:** Fewer parameters and better generalization by learning time-invariant patterns.

**Limitation:** It cannot model time-specific behavior since all steps use the same shared weights

**2) Vanishing Gradients & Remedies**

a) Describe the vanishing gradient problem in RNNs and how it affects long-range dependencies.

Ans: Vanishing gradients occur when gradients shrink exponentially during backpropagation through many time steps, causing the RNN to fail to learn long-range dependencies because early time-step weights receive almost no gradient signal.

b) List two architectural solutions and briefly how each helps gradient flow.

Ans: **LSTM:** Uses gates and a cell state that provides a constant error path, preventing gradients from shrinking over long sequences.

**GRU:** Uses update/reset gates that regulate information flow, helping preserve gradients across time.

c) Give one training technique (not an architecture change) that can mitigate the issue and why.

Ans: **Gradient clipping:** Prevents exploding/unstable updates, stabilizing backpropagation and helping gradients stay in a usable range over long sequences.

**3) LSTM Gates & Cell State**

a) Explain the roles of the forget, input, and output gates. For each, name its activation and purpose.

Ans: **Forget gate** — σ (sigmoid): decides what portion of the previous cell state to erase.

**Input gate** — σ (sigmoid) + tanh: decides which new information to add to the cell state.

**Output gate** — σ (sigmoid) + tanh: controls how much of the cell state is revealed as the hidden state/output.

b) Why is the LSTM cell state often described as providing a "linear path" for gradients?

Ans: The LSTM cell state updates with element-wise additions and scaling instead of repeated multiplications, allowing gradients to pass through many time steps without vanishing.

c) In one or two sentences, contrast "what to remember" vs. "what to expose" in LSTMs.

Ans: **"What to remember"- is controlled by the forget and input gates, which decide how the cell state is updated.**

**"What to expose" - is controlled by the output gate, which determines how much of the internal memory becomes visible as the hidden state.**

**4) Self-Attention**

a) Define Query (Q), Key (K), and Value (V) in the context of self-attention.

Ans: **Query (Q):** a vector representing *what a token is looking for* in other tokens.

**Key (K):** a vector representing *what each token offers* or how it should be matched against queries.

**Value (V):** a vector containing the *actual information/content* to be aggregated after attention scores are computed.

b) Write the formula for dot-product attention.

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

c) Why do we divide by $\sqrt{d_k}$?

Ans: To prevent the dot products from growing too large as dimensionality increases, which keeps the softmax stable and avoids extremely small gradients.

**5) Multi-Head Attention & Residual Connections**

a) Why do Transformers use multi-head attention instead of single-head attention?

Ans: Multi-head attention lets the model attend to different types of relationships (positions, syntax, meaning) in parallel, giving richer representations than a single-head can capture.

b) What is the purpose of Add & Norm (Residual + LayerNorm)? Explain two benefits.

Ans: **Residual** connections help preserve gradients and prevent degradation when stacking many layers.

**LayerNorm** stabilizes activations and speeds up training by normalizing each layer's outputs.

c) Describe one example of linguistic relation that different heads might capture (e.g., coreference, syntax).

Ans: One head might track coreference (e.g., "John… he"), while another tracks syntactic dependencies like subject–verb or adjective–noun relationships.

**6) Encoder–Decoder with Masked Attention**

a) Why does the decoder use masked self-attention? What problem does it prevent?

Ans: To prevent the decoder from looking at future tokens, ensuring it only predicts based on past outputs and avoiding information leakage.

b) What is the difference between encoder self-attention and encoder–decoder cross-attention?

Ans: **Encoder self-attention:** tokens attend only to other input tokens.

**Encoder–decoder cross-attention:** decoder queries attend to encoder outputs to use source-sentence information.

c) During inference (no teacher forcing), how does the model generate tokens step by step?

Ans: The model generates one token at a time, feeds it back into the decoder, and repeats the process until an end-of-sentence token is produced.

**Part II: Programming**

**Q1. Character-Level RNN Language Model ("hello" toy & beyond)**
Goal: Train a tiny character-level RNN to predict the next character given previous characters.

Data (toy to start):

- Start with a small toy corpus you create (e.g., several "hello…", "help…", short words/sentences).
- Then expand to a short plain-text file of ~50–200 KB (any public-domain text of your choice).

Model:

- Embedding → RNN (Vanilla RNN or GRU or LSTM) → Linear → Softmax over characters.

- Hidden size 64–256; sequence length 50–100; batch size 64; train 5–20 epochs.

Train:

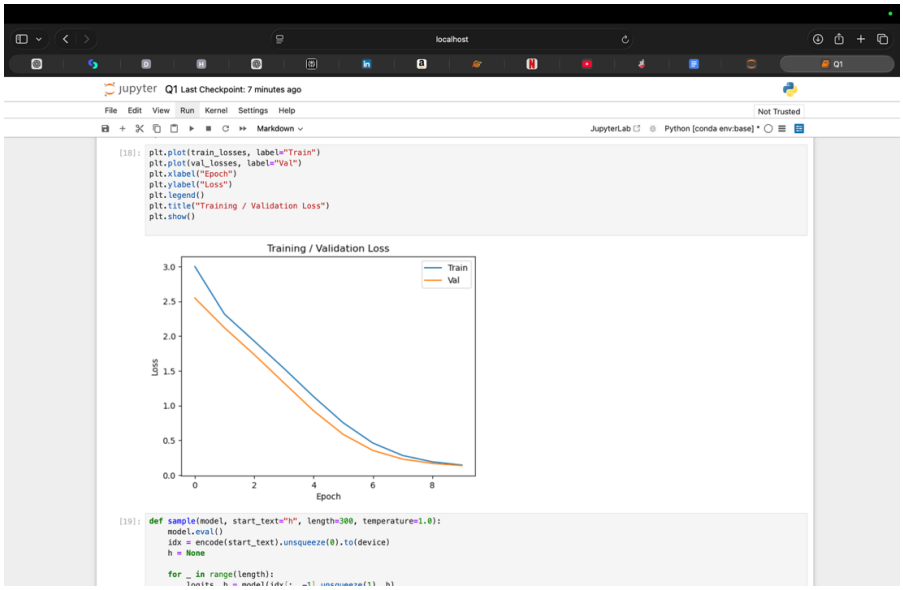
- Teacher forcing (use the true previous char as input during training).
- Cross-entropy loss; Adam optimizer.

Report:

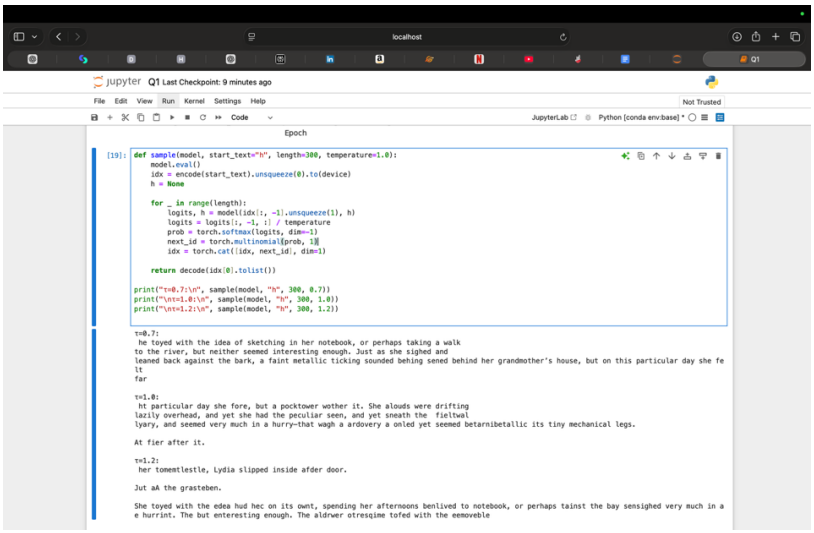
1. Training/validation loss curves.
2. Sample 3 temperature-controlled generations (e.g., $\tau = 0.7, 1.0, 1.2$) for 200–400 chars each.
3. A 3–5 sentence reflection: what changes when you vary sequence length, hidden size, and temperature?
   (Connect to slides: embedding, sampling loop, teacher forcing, tradeoffs

Ans:

1.



2.




3. Increasing the sequence length allows the model to capture longer-range dependencies, improving coherence but also increasing training time and difficulty. A larger hidden size usually improves the model's capacity to learn richer patterns, but it may overfit and requires more memory. During sampling, temperature strongly affects creativity: low $\tau$ ($\approx$0.7) produces more repetitive and conservative text, $\tau = 1.0$ gives balanced diversity, and high $\tau$ (1.2+) produces creative but often incoherent sequences. Teacher forcing helps the model converge faster by providing the correct previous character, but it can make sampling more brittle when the model must rely on its own predictions. Overall, the RNN learns meaningful character-level structure, but its generation quality is sensitive to these hyperparameters.

## Q2. Mini Transformer Encoder for Sentences

Task: Build a mini Transformer Encoder (NOT full decoder) to process a batch of sentences.

Steps:

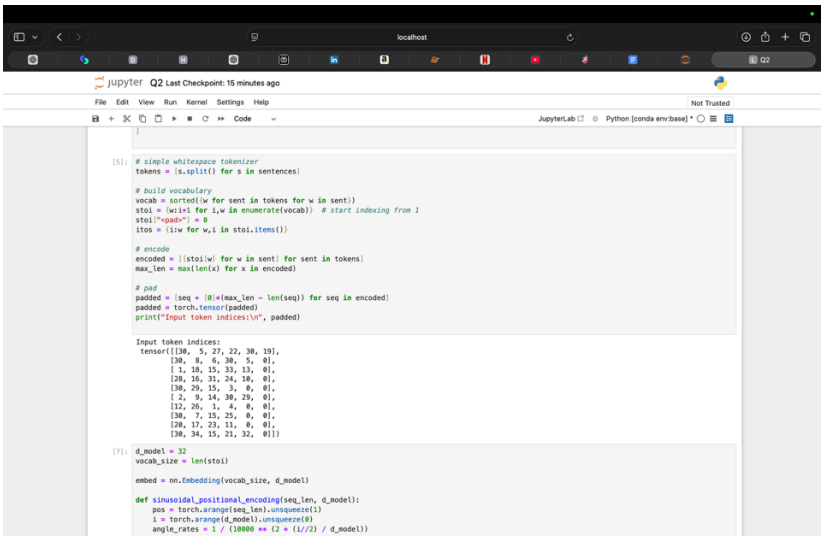
1. Use a small dataset (e.g., 10 short sentences of your choice).
2. Tokenize and embed the text.
3. Add sinusoidal positional encoding.
4. Implement:
    - Self-attention layer
    - Multi-head attention (2 or 4 heads)
    - Feed-forward layer

- Add & Norm

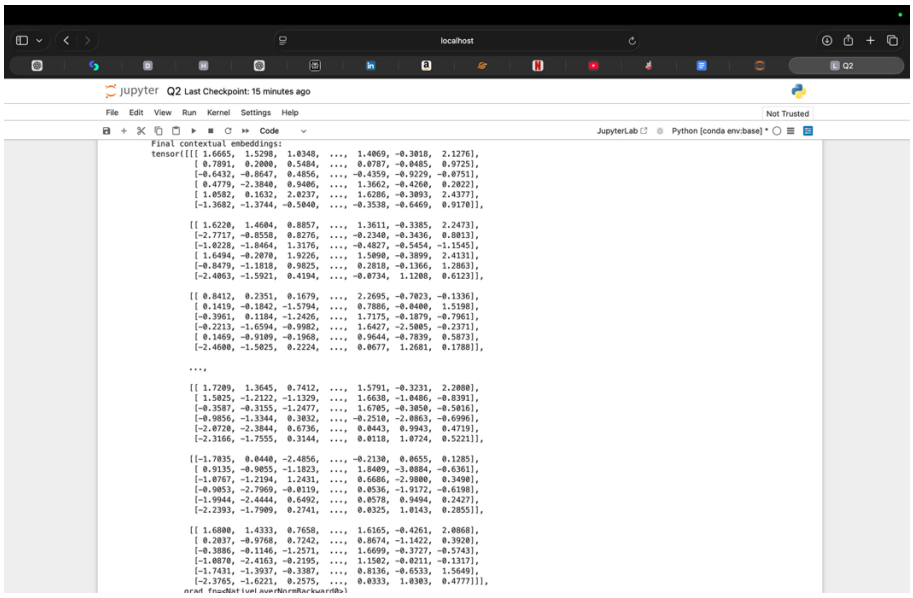
5. Show:
   - Input tokens
   - Final contextual embeddings
   - Attention heatmap between words (visual or printed)

Ans:

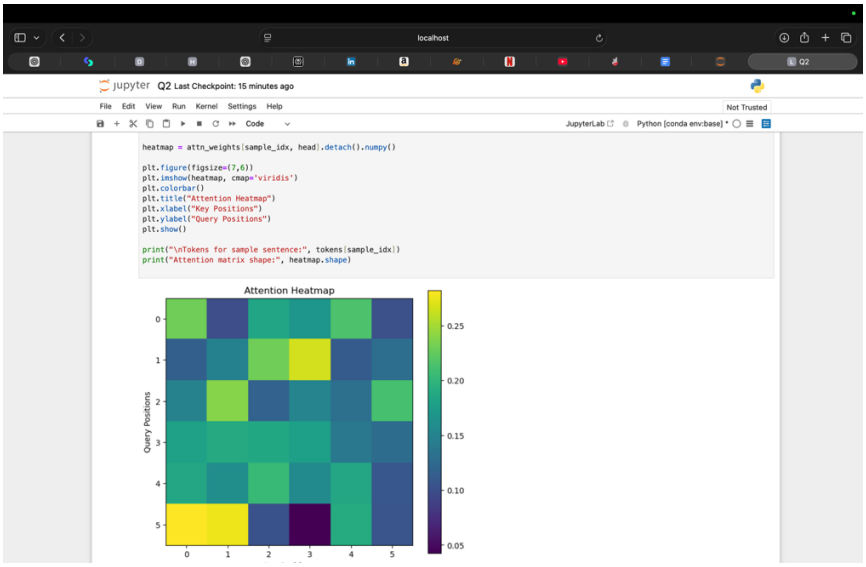
1.




2.



3.




## Q3. Implement Scaled Dot-Product Attention

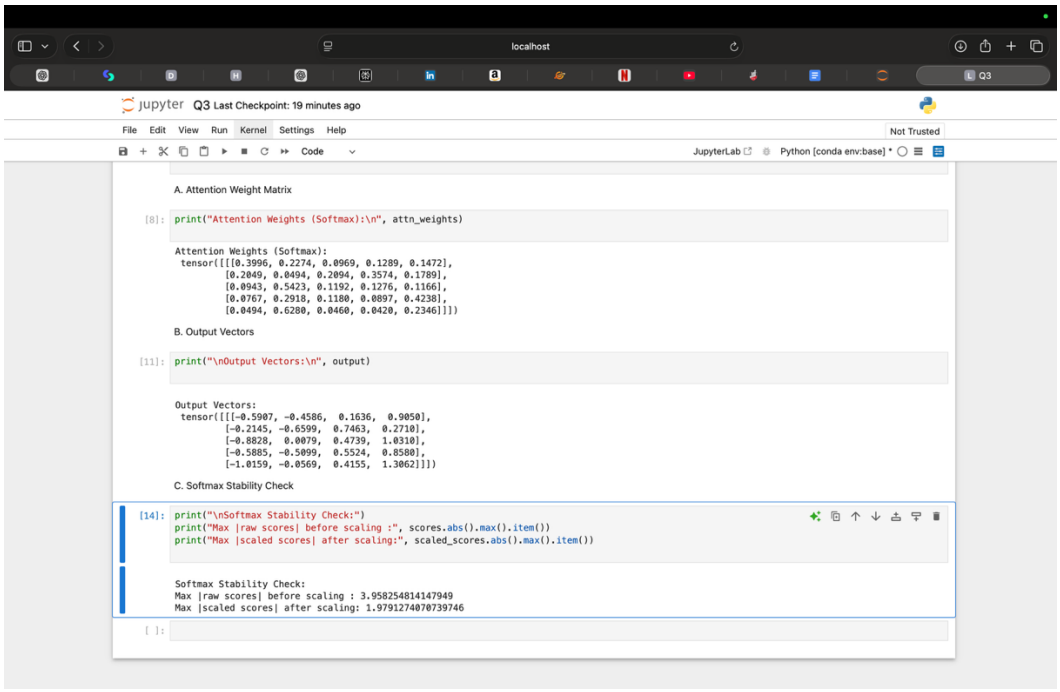
Goal: Implement the attention function from your slides:

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

Requirements:

- Write a function in PyTorch or TensorFlow to compute attention.
- Test it using random Q, K, V inputs.
- Print:
  - Attention weight matrix
  - Output vectors
  - Softmax stability check (before and after scaling)

Ans:


localhost
jupyter Q3 Last Checkpoint: 19 minutes ago
File Edit View Run Kernel Settings Help
Not Trusted
Code
JupyterLab
Python [conda env:base]
A. Attention Weight Matrix
[8]: print("Attention Weights (Softmax):\n", attn_weights)
Attention Weights (Softmax):
tensor([[[0.3996, 0.2274, 0.0969, 0.1289, 0.1472],
[0.2049, 0.0494, 0.2094, 0.3574, 0.1789],
[0.0943, 0.5423, 0.1192, 0.1276, 0.1166],
[0.0767, 0.2918, 0.1180, 0.0897, 0.4238],
[0.0494, 0.6280, 0.0460, 0.0420, 0.2346]]])
B. Output Vectors
[11]: print("\nOutput Vectors:\n", output)
Output Vectors:
tensor([[[-0.5907, -0.4586, 0.1636, 0.9050],
[-0.2145, -0.6599, 0.7463, 0.2710],
[-0.8828, 0.0079, 0.4739, 1.0310],
[-0.5885, -0.5099, 0.5524, 0.8580],
[-1.0159, -0.0569, 0.4155, 1.3062]]])
C. Softmax Stability Check
[14]: print("\nSoftmax Stability Check:")
print("Max |raw scores| before scaling :", scores.abs().max().item())
print("Max |scaled scores| after scaling:", scaled_scores.abs().max().item())
Softmax Stability Check:
Max |raw scores| before scaling : 3.958254814147949
Max |scaled scores| after scaling: 1.979127407073974
[ ]: