University of Central Missouri

Department of Computer Science & Cybersecurity

CS5760 Natural Language Processing

Fall 2025

Homework 4.

Student name: SANTHOSH REDDY KISTIPATI

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.

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

Ans : **One-to-Many**

Because you start with a single input (like a start token or context vector) and the model generates a *sequence* of predicted words step by step

• sentiment of a sentence

Ans : **Sentiment of a sentence → Many-to-One**

The model reads the full sequence and outputs *one* label — the overall sentiment — so collapsing all time-steps into a single decision.

• NER

Ans : **Many-to-Many (Aligned)**

NER needs one label per word, so each input word maps directly to an output tag at the same timestep.

• machine translation

Ans : **Many-to-Many (Unaligned)**

The input sequence is encoded first, and the decoder generates an output sequence of *different length* with no one-to-one alignment.

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

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

Ans : Unrolling copies the same RNN cell across all timesteps so that BPTT can compute gradients through each step while sharing the same weights at every position.

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

Ans : **Advantage:**
Weight sharing reduces the number of parameters, letting the model generalize better and learn consistent patterns across time instead of memorizing step-specific transformations.

Limitation:
Since the same weights are reused at every timestep, the model cannot specialize its behavior for different positions, making it harder to capture long-range dependencies or encode time-specific information.

2) Vanishing Gradients & Remedies
a) Describe the vanishing gradient problem in RNNs and how it affects long-range dependencies.

Ans 👍The vanishing gradient problem occurs when gradients shrink exponentially as they are backpropagated through many timesteps in an RNN, causing earlier layers to receive almost no learning signal. Because of this, the model essentially "forgets" information from far-back timesteps — it cannot connect distant causes and effects — so long-range dependencies collapse and the RNN behaves like it only remembers the most recent inputs.

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

Ans 👍 **LSTMs:**
LSTMs use gated memory cells that create near-linear gradient paths, letting error signals bypass repeated nonlinear squashing. This prevents gradients from dying out and preserves information across long sequences.

• GRUs:
GRUs use update and reset gates to control how much past information flows forward. These gates allow gradients to pass through fewer transformations, making the flow smoother and reducing vanishing compared to vanilla RNNs.

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

Ans 👍**Gradient clipping** helps by preventing exploding gradients from destabilizing learning, which indirectly keeps the optimization landscape stable and prevents the model from being

pushed into regions where vanishing gradients worsen. By keeping gradient magnitudes controlled, the network maintains healthier updates across timesteps.

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 (sigmoid):**
This gate uses a sigmoid function to decide which parts of the previous cell state to erase. Its purpose is to flush outdated or irrelevant information so the cell doesn't carry junk forward.

Input Gate (sigmoid + tanh):
The sigmoid part chooses *which new information is allowed in*, and the tanh part creates candidate values to be added. Together, they control what new content gets written into long-term memory.

Output Gate (sigmoid + tanh):
The sigmoid gate decides how much of the cell state should influence the hidden state, while tanh transforms the cell state into a bounded form. Its purpose is to control what information is *revealed* to the next layer or timestep.

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

Ans 👍The LSTM cell state updates through mostly additive operations instead of repeated multiplications, which means gradients travel through it without being squashed by nonlinearities at every step. This near-linear flow preserves gradient magnitude across long sequences and prevents 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 what stays in long-term memory. "What to expose" is controlled by the output gate, which decides what part of that stored information becomes visible in the hidden state at the current timestep.

4) Self-Attention

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

Ans 👍In self-attention, each token is projected into three different vectors: the **Query** represents what the current token is *looking for*, the **Key** represents what each token *offers* for comparison, and the **Value** holds the actual information that gets mixed into the final output after attention weights are applied.

b) Write the formula for dot-product attention.

Ans 👍Attention(Q, K, V) = softmax((QK^T) / sqrt(d_k)) * V

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

Ans 👍 Because without scaling, the dot products grow too large as the dimension increases, pushing the softmax into extremely sharp distributions. Dividing by $\sqrt{d_k}$ keeps the scores in a stable range, prevents exploding logits, and ensures smoother gradients during training.

5) Multi-Head Attention & Residual Connections

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

Ans 👍Transformers use multi-head attention because multiple heads let the model focus on different types of relationships at the same time—one head might track long-range dependencies while another picks up local patterns. A single head would collapse everything into one similarity space, losing these diverse perspectives.

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

Ans 👍Residual connections let the model pass information forward unchanged, preventing deeper layers from “forgetting” useful signals and avoiding vanishing gradients. Layer normalization then stabilizes the activations by keeping them on a consistent scale, which makes training faster and prevents unstable updates as the network grows deeper.

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

Ans 👍Different heads can specialize—for example, one head may track **coreference** (linking a pronoun to the noun it refers to), while another may focus on **syntactic structure** like subject–verb alignment or modifier relationships.

6) Encoder–Decoder with Masked Attention

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

Ans 👍The decoder uses masked self-attention to block access to future tokens, ensuring each position only attends to current and previous words. This prevents the model from "cheating" during training by looking ahead, which would break the autoregressive requirement and make generation logically impossible at test time.

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

Ans 👍Encoder self-attention lets every input token attend to all other input tokens, building a contextualized representation of the entire source sentence. Cross-attention happens in the decoder, where each decoder token attends to the encoder's outputs, allowing the decoder to pull in relevant information from the source sequence while generating the translation.

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

Ans 👍During inference, the model predicts one token, appends that predicted token to the input sequence, feeds the whole growing sequence back into the decoder, and then repeats this process. Each step depends on all previously generated outputs, so generation proceeds strictly left-to-right.

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

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)

Q3. Implement Scaled Dot-Product Attention

Goal: Implement the attention function from your slides:

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

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)