MIDDLE EAST TECHNICAL UNIVERSITY

SEMESTER I EXAMINATION 2024-2025

CENG 403 – Deep Learning - RNNs, LSTM & Language Models (University Sources) - ANSWERED

January 2025 TIME ALLOWED: 3 HOURS

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **SEVEN (7)** questions and comprises **TEN (10)** printed pages.
2. Answer all questions. The marks for each question are indicated at the beginning of each question.
3. Answer each question beginning on a **FRESH** page of the answer book.
4. This **IS NOT an OPEN BOOK** exam.
5. Show all mathematical derivations clearly with proper notation.
6. For implementation questions, provide clear pseudocode or algorithms.
7. Explain computational complexity where requested.
8. Draw clear architectural diagrams with proper labels.

Question 1. Backpropagation Through Time (BPTT) (25 marks)
Based on Stanford/MIT/University of Toronto deep learning courses.

(a) Define Backpropagation Through Time (BPTT) and explain how it extends traditional backpropagation to sequential data. Include the mathematical formulation for unfolding an RNN across time steps. (8 marks)

Answer: BPTT is the algorithm for training RNNs by unfolding the recurrent network through time and applying backpropagation to the unfolded computational graph.

Definition and Core Concept: BPTT extends traditional backpropagation to handle sequential data by "unfolding" the RNN through time. For a sequence of length T, we create T copies of the RNN cell, each processing one time step.

Mathematical Formulation: For an RNN with hidden state $h_t = f(h_{t-1}, x_t; \theta)$, we unfold it as:

$$h_1 = f(h_0, x_1; \theta) \tag{1}$$
$$h_2 = f(h_1, x_2; \theta) \tag{2}$$
$$\vdots \tag{3}$$
$$h_T = f(h_{T-1}, x_T; \theta) \tag{4}$$

The total loss is: $L = \sum_{t=1}^{T} L_t(h_t, y_t)$

Key Differences from Standard Backpropagation:

- Parameters are shared across time steps
- Gradients flow both backwards through layers and backwards through time
- Total gradient for shared parameters is the sum across all time steps

(b) Consider an RNN with hidden state update equation: (12 marks)

$$h_t = \tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

For a sequence of length T=3, derive the gradient $\frac{\partial L}{\partial W_{hh}}$ where $L = \sum_{t=1}^{3} L_t$ is the total loss. Show the complete gradient computation including the chain rule applications.

Answer: The gradient $\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{3} \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial h_k} \cdot \frac{\partial h_k}{\partial W_{hh}}$

Step-by-Step Derivation:

Since parameters are shared, we need to sum gradients from all time steps where W_{hh} affects the loss:

1. Direct Dependencies:

$$\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{3} \frac{\partial L_t}{\partial W_{hh}} \tag{5}$$

2. For each time step t, W_{hh} affects L_t through all previous hidden states:

$$\frac{\partial L_t}{\partial W_{hh}} = \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \cdot \frac{\partial h_t}{\partial h_k} \cdot \frac{\partial h_k}{\partial W_{hh}} \tag{6}$$

3. Computing the components: $\frac{\partial h_k}{\partial W_{hh}} = (1-\tanh^2(W_{hx}x_k + W_{hh}h_{k-1} + b_h)) \cdot h_{k-1}$

$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} (1 - \tanh^2(\cdot)) \cdot W_{hh}$

4. Complete gradient for T=3:

$$\frac{\partial L}{\partial W_{hh}} = \frac{\partial L_1}{\partial h_1} \cdot (1 - \tanh^2(\cdot))_1 \cdot h_0 \tag{7}$$

$$+ \frac{\partial L_2}{\partial h_2} \cdot (1 - \tanh^2(\cdot))_2 \cdot h_1 \tag{8}$$

$$+ \frac{\partial L_2}{\partial h_2} \cdot (1 - \tanh^2(\cdot))_2 \cdot W_{hh} \cdot (1 - \tanh^2(\cdot))_1 \cdot h_0 \tag{9}$$

$$+ \frac{\partial L_3}{\partial h_3} \cdot (1 - \tanh^2(\cdot))_3 \cdot h_2 \tag{10}$$

$$+ \text{terms for } h_3 \text{ dependence on } h_1 \text{ and } h_0 \tag{11}$$

(c) Explain truncated BPTT and why it's necessary for long sequences. Compare regular truncation vs. randomized truncation approaches, discussing computational complexity and numerical stability. (5 marks)

Answer: Truncated BPTT limits the number of time steps for gradient computation to prevent vanishing gradients and reduce computational cost in long sequences.

Why Truncated BPTT is Necessary:

- **Computational Cost:** BPTT complexity is $O(T^2)$ for sequence length T
- **Memory Requirements:** Must store all intermediate states
- **Vanishing Gradients:** Gradients decay exponentially with sequence length
- **Numerical Instability:** Long gradient paths cause numerical issues

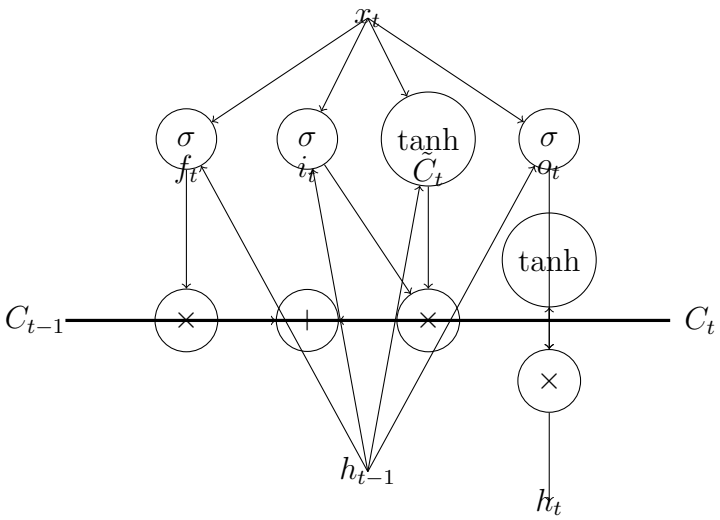
Regular Truncation: - Fixed window size k (e.g., k=20) - Gradients computed only k steps back - Complexity: O(k) instead of O(T) - Predictable memory usage

Randomized Truncation: - Random truncation length at each step - Unbiased gradient estimation in expectation - Better handling of different time dependencies - More robust to sequence length variations

Trade-offs: Regular truncation is simpler but may miss long-term dependencies. Randomized truncation provides better theoretical guarantees but is more complex to implement.

Question 2. LSTM Architecture and Gating Mechanisms (30 marks)
Based on university deep learning course materials and D2L.ai educational content.

(a) Draw the complete LSTM cell architecture showing all gates (input, forget, output) and state components (cell state, hidden state). Label all weight matrices and explain information flow. (10 marks)



Answer: The LSTM architecture consists of three gates (forget, input, output) and two state vectors (cell state, hidden state) with specific weight matrices for each component.

Components and Information Flow:

- **Forget Gate (f_t):** Controls what information to discard from cell state
- **Input Gate (i_t):** Controls what new information to store in cell state
- **Candidate Values ($\tilde{C}_t$):** New candidate values for cell state
- **Output Gate (o_t):** Controls what parts of cell state to output
- **Cell State (C_t):** Long-term memory, flows horizontally

- **Hidden State (h_t):** Short-term memory, filtered cell state

Weight Matrices: Each gate has input weights W and recurrent weights U: W_f, W_i, W_C, W_o for inputs and U_f, U_i, U_C, U_o for hidden states.

(b) Write the mathematical equations for all LSTM gates and state updates. Given input x_t, previous hidden state h_{t-1}, and previous cell state C_{t-1}, derive: (15 marks)

- Forget gate: f_t =?
- Input gate: i_t =?
- Candidate values: $\tilde{C}_t$ =?
- Cell state update: C_t =?
- Output gate: o_t =?
- Hidden state: h_t =?

Answer: Complete LSTM equations with proper weight matrices and activation functions:

LSTM Mathematical Formulation:

1. Forget Gate: Controls what to forget from previous cell state

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

2. Input Gate: Controls what new information to store

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

3. Candidate Values: New candidate values to potentially store

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

4. Cell State Update: Combines forgotten old state with new candidates

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

5. Output Gate: Controls what parts of cell state to output

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

6. Hidden State: Filtered version of cell state

$$h_t = o_t \odot \tanh(C_t)$$

Key Insights:

- σ is the sigmoid function (outputs 0-1 for gating)
- tanh normalizes values to [-1, 1]
- $\odot$ denotes element-wise multiplication
- $[h_{t-1}, x_t]$ represents concatenation of vectors

(c) Compare LSTM vs. vanilla RNN in terms of gradient flow. Explain how the cell state pathway in LSTM addresses the vanishing gradient problem through mathematical analysis of gradient propagation. (5 marks)

Answer: LSTM solves vanishing gradients through the cell state pathway, which provides a highway for gradient flow with minimal transformations.

Vanilla RNN Gradient Flow: For vanilla RNN: $h_t = \tanh(Wh_{t-1} + Ux_t)$

Gradient through time: $\frac{\partial h_t}{\partial h_{t-k}} = \prod_{i=1}^{k} \frac{\partial h_{t-i+1}}{\partial h_{t-i}} = \prod_{i=1}^{k} W \cdot \text{diag}(\tanh'(\cdot))$

Since $|\tanh'(x)| \leq 1$ and typically $<< 1$, gradients vanish exponentially.

LSTM Cell State Gradient Flow: The cell state update: $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$

Gradient: $\frac{\partial C_t}{\partial C_{t-1}} = f_t$ (element-wise)

Key Advantages:

- No matrix multiplication (unlike vanilla RNN's W)
- Forget gate values can be close to 1, preserving gradients
- Additive updates prevent multiplicative vanishing
- Direct gradient pathway from C_t to C_{t-k} through forget gates

This creates a "gradient highway" that maintains gradient flow across long sequences.

Question 3. Word Embeddings and Distributional Semantics (22 marks)
Based on NLP course materials from Stanford CS224n and similar university programs.

(a) Explain the distributional hypothesis that underlies word embeddings. How does the CBOW (Continuous Bag of Words) model implement this principle? (6 marks)

Answer: The distributional hypothesis states that words appearing in similar contexts have similar meanings. CBOW implements this by predicting a target word from its context words.

Distributional Hypothesis: "You shall know a word by the company it keeps" - words that appear in similar contexts tend to have similar meanings. This is the foundation of distributional semantics.

CBOW Implementation:

- **Input:** Context words around target word (e.g., window size = 2)
- **Goal:** Predict the center word from context
- **Architecture:**
  (a) One-hot encode context words
  (b) Average their embeddings (bag of words)
  (c) Use averaged embedding to predict target word
- **Learning:** Words with similar contexts get similar embeddings because they're used to predict similar target words

Mathematical Formulation: Given context words $w_{t-c}, ..., w_{t-1}, w_{t+1}, ..., w_{t+c}$:

$$h = \frac{1}{2c} \sum_{j \neq 0, |j| \leq c} W_{embed} \cdot e_{t+j}$$

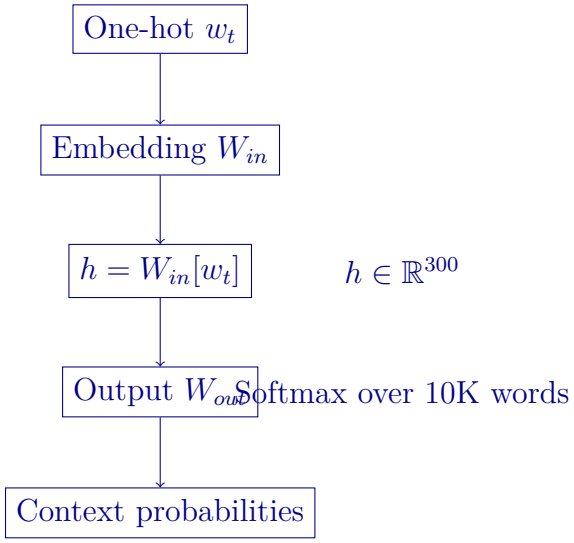
$$P(w_t | context) = \text{softmax}(W_{out} \cdot h)$$

(b) Design the Skip-gram model architecture for learning word embeddings. Given vocabulary size V=10,000 and embedding dimension d=300: (10 marks)

- Draw the network architecture
- Calculate the number of parameters
- Explain the softmax bottleneck and hierarchical softmax solution
- Derive the loss function using negative sampling

Answer: Skip-gram predicts context words from target word, with input embedding matrix W_{in} (V×d) and output matrix W_{out} (d×V).

Architecture:



Parameter Count:

- Input embedding matrix: $W_{in} = 10,000 \times 300 = 3,000,000$
- Output weight matrix: $W_{out} = 300 \times 10,000 = 3,000,000$
- Total parameters: $6,000,000$

Softmax Bottleneck: Computing softmax over 10K words is expensive: $O(V \cdot d)$ per prediction.

Hierarchical Softmax: Uses binary tree structure where each word is a leaf. Path from root to leaf defines probability:

$$P(w|context) = \prod_{i=1}^{L(w)} \sigma([[n(w, i+1) = \text{ch}(n(w, i))]] \cdot h^T \theta_{n(w,i)})$$

Reduces complexity from $O(V)$ to $O(\log V)$.

Negative Sampling Loss: For target word w_t and context w_c:

$$L = -\log \sigma(u_{w_c}^T v_{w_t}) - \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[\log \sigma(-u_{w_i}^T v_{w_t})]$$

where k is number of negative samples and $P_n(w)$ is noise distribution.

(c) Analyze word embedding arithmetic: "king - man + woman queen". Explain: (6 marks)

- Why this works mathematically in embedding space
- What linguistic relationships are captured
- Limitations of linear analogies in embeddings

Answer: Word arithmetic works because embeddings capture linear relationships between semantic concepts through vector differences.

Mathematical Explanation: The embedding space captures relational similarities. If we denote the relationship "male to female" as a vector $\vec{r}$, then:

$$\vec{king} - \vec{man} \approx \vec{r}_{royalty} \tag{12}$$

$$\vec{woman} + \vec{r}_{royalty} \approx \vec{queen} \tag{13}$$

The vector difference $\vec{king} - \vec{man}$ captures the "royalty" relationship, which when added to $\vec{woman}$ gives the female royal equivalent.

Linguistic Relationships Captured:

- **Gender:** man/woman, king/queen, actor/actress
- **Tense:** walk/walked, go/went
- **Comparative:** good/better, big/bigger
- **Plural:** car/cars, child/children
- **Geography:** Paris/France, Tokyo/Japan

Limitations:

- **Polysemy:** Multiple word meanings not well handled

- **Complex Relations:** Non-linear relationships fail
- **Cultural Bias:** Embeddings inherit training data biases
- **Compositionality:** Phrase meanings don't always compose linearly
- **Context Independence:** Same embedding regardless of context

Question 4. Sequence-to-Sequence Models and Attention (28 marks)
Based on modern NLP course materials covering encoder-decoder architectures.

(a) Design an encoder-decoder RNN architecture for machine translation from English to French. Show the complete architecture including: (12 marks)

- Encoder RNN processing input sequence
- Context vector computation
- Decoder RNN generating output sequence
- How teacher forcing works during training
- Inference procedure using greedy/beam search

Answer: Encoder-decoder architecture uses two RNNs: encoder processes input sequence to create context vector, decoder generates output sequence conditioned on this context.

Complete Architecture:

1. Encoder RNN:

- Input: English sentence $x_1, x_2, ..., x_T$
- Hidden states: $h_1^{(enc)}, h_2^{(enc)}, ..., h_T^{(enc)}$
- Equations: $h_t^{(enc)} = f_{enc}(h_{t-1}^{(enc)}, x_t)$

2. Context Vector:

- Simple approach: $c = h_T^{(enc)}$ (final encoder state)
- Alternative: $c = \tanh(W_c h_T^{(enc)})$ (learned transformation)

3. Decoder RNN:

- Initial state: $h_0^{(dec)} = c$
- At each step: $h_t^{(dec)} = f_{dec}(h_{t-1}^{(dec)}, y_{t-1})$
- Output: $P(y_t) = \text{softmax}(W_o h_t^{(dec)})$

4. Teacher Forcing (Training):

- Use ground truth y_{t-1} as input to decoder at step t
- Speeds up training and provides stable gradients
- Loss: $L = -\sum_{t=1}^{T'} \log P(y_t^* | y_1^*, ..., y_{t-1}^*, x)$

5. Inference:

- **Greedy:** Select highest probability word at each step
- **Beam Search:** Maintain top-k sequences, expand each
- Handle <EOS> token to terminate sequences

(b) Implement beam search decoding with beam size k=3. Given the following probability distributions over vocabulary A, B, C, <EOS> for 2 time steps: (10 marks)

Context	P(A)	P(B)	P(C)	P(<EOS>)
Initial	0.5	0.3	0.15	0.05
After A	0.2	0.1	0.6	0.1
After B	0.4	0.2	0.3	0.1
After C	0.1	0.7	0.1	0.1

Show the complete beam search tree and final ranked sequences.

Answer: Beam search maintains top-3 sequences at each step, expanding each and keeping the best candidates.

Step 1: Initial Expansion From <START>, expand to all tokens:

- A: score = 0.5
- B: score = 0.3
- C: score = 0.15

Keep top 3: [A: 0.5], [B: 0.3], [C: 0.15]

Step 2: Expand Each Beam

From A (score: 0.5):

- AA: $0.5 \times 0.2 = 0.10$

- AB: $0.5 \times 0.1 = 0.05$
- AC: $0.5 \times 0.6 = 0.30$
- A<EOS>: $0.5 \times 0.1 = 0.05$

From B (score: 0.3):

- BA: $0.3 \times 0.4 = 0.12$
- BB: $0.3 \times 0.2 = 0.06$
- BC: $0.3 \times 0.3 = 0.09$
- B<EOS>: $0.3 \times 0.1 = 0.03$

From C (score: 0.15):

- CA: $0.15 \times 0.1 = 0.015$
- CB: $0.15 \times 0.7 = 0.105$
- CC: $0.15 \times 0.1 = 0.015$
- C<EOS>: $0.15 \times 0.1 = 0.015$

Final Ranking (Top 3):

(a) AC: 0.30

(b) BA: 0.12

(c) CB: 0.105

(c) Explain the attention mechanism as a solution to the bottleneck problem in sequence-to-sequence models. Derive the mathematical formulation for Bahdanau attention. (6 marks)

Answer: Attention solves the bottleneck by allowing decoder to access all encoder states, not just the final context vector.

Bottleneck Problem: Standard seq2seq compresses entire input sequence into fixed-size context vector $c = h_T^{(enc)}$. This creates information loss for long sequences and makes it difficult to align input-output positions.

Attention Solution: Instead of single context vector, compute different context vector c_t at each decoder step based on all encoder states.

Bahdanau Attention Formulation:

1. Attention Scores:

$$e_{t,j} = a(h_{t-1}^{(dec)}, h_j^{(enc)})$$

where a is alignment model: $a(h_{t-1}^{(dec)}, h_j^{(enc)}) = v_a^T \tanh(W_a h_{t-1}^{(dec)} + U_a h_j^{(enc)})$

2. Attention Weights:

$$\alpha_{t,j} = \frac{\exp(e_{t,j})}{\sum_{k=1}^{T} \exp(e_{t,k})}$$

3. Context Vector:

$$c_t = \sum_{j=1}^{T} \alpha_{t,j} h_j^{(enc)}$$

4. Decoder Update:

$$h_t^{(dec)} = f_{dec}(h_{t-1}^{(dec)}, y_{t-1}, c_t)$$

Key Benefits:

- No information bottleneck
- Automatic alignment learning
- Interpretable attention weights
- Better handling of long sequences

Question 5. Gradient Problems and Solutions (20 marks)
Based on theoretical analysis from university deep learning courses.

(a) Analyze the vanishing gradient problem in RNNs. For an RNN with hidden state transition $h_t = \tanh(Wh_{t-1} + Ux_t)$, show mathematically why gradients vanish for long sequences. (8 marks)

Include analysis of:

- Gradient computation through multiple time steps
- Effect of tanh derivative bounds
- Impact of weight matrix eigenvalues

Answer: Gradients vanish due to repeated multiplication of small derivatives and weight matrices with eigenvalues less than 1.

Gradient Computation Through Time:

For loss at time t, gradient w.r.t. hidden state at time $t-k$:

$$\frac{\partial L_t}{\partial h_{t-k}} = \frac{\partial L_t}{\partial h_t} \prod_{i=1}^{k} \frac{\partial h_{t-i+1}}{\partial h_{t-i}}$$

Individual Gradient Term:

$$\frac{\partial h_{t-i+1}}{\partial h_{t-i}} = \frac{\partial \tanh(Wh_{t-i} + Ux_{t-i+1})}{\partial h_{t-i}} = W \cdot \text{diag}(\tanh'(Wh_{t-i} + Ux_{t-i+1}))$$

Effect of Tanh Derivative: Since $\tanh'(x) = 1 - \tanh^2(x)$ and $|\tanh(x)| \leq 1$:

- $\tanh'(x) \in [0, 1]$
- For saturated regions: $\tanh'(x) \approx 0$
- Maximum at $x = 0$: $\tanh'(0) = 1$

Complete Gradient:

$$\frac{\partial L_t}{\partial h_{t-k}} = \frac{\partial L_t}{\partial h_t} \prod_{i=1}^{k} W \cdot \text{diag}(\tanh'(\cdot))$$

Why Gradients Vanish:

- **Derivative bounds:** Each $\tanh'(\cdot) \leq 1$, typically much smaller
- **Matrix multiplication:** If largest eigenvalue $\lambda_{max}(W) < 1$, then $\|W^k\|$ decays exponentially
- **Combined effect:** Product of k terms, each ≤ 1, causes exponential decay

Mathematical Bound:

$$\left\| \frac{\partial L_t}{\partial h_{t-k}} \right\| \leq \left\| \frac{\partial L_t}{\partial h_t} \right\| \cdot \gamma^k$$

where $\gamma = \lambda_{max}(W) \cdot \max_i(\tanh'(\cdot))$ and typically $\gamma < 1$.

(b) Compare three solutions to gradient problems in RNNs: (12 marks)

- Gradient clipping (explain algorithm and threshold selection)
- LSTM gating mechanisms (focus on gradient flow)
- Skip connections in deep RNNs

Provide mathematical justification for each approach.

Answer: Three complementary solutions: gradient clipping prevents explosion, LSTM gates enable selective flow, skip connections provide direct paths.

1. Gradient Clipping:

Algorithm:

Compute gradient $g = \nabla_\theta L$
Compute gradient norm $\|g\|$
if $\|g\| >$ threshold **then**
  $g \leftarrow \frac{\text{threshold}}{\|g\|} \cdot g$
end if
Update parameters: $\theta \leftarrow \theta - \alpha g$

Threshold Selection:

- Monitor gradient norms during training
- Choose threshold as 95th percentile of observed norms
- Typical values: 1.0-5.0 for RNNs

Mathematical Justification: Prevents parameter updates that are too large: $\|\Delta\theta\| = \alpha\|g\| \leq \alpha \cdot \text{threshold}$

2. LSTM Gating Mechanisms:

Gradient Flow Analysis: Cell state gradient: $\frac{\partial C_t}{\partial C_{t-1}} = f_t$

Unlike vanilla RNN, no matrix multiplication or bounded activation in the direct path.

Key Properties:

- **Forget gate values:** Can be close to 1, preserving gradients
- **Additive updates:** $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$
- **Selective information flow:** Gates learn when to preserve/update

3. Skip Connections in Deep RNNs:

Architecture:

$$h_t^{(l)} = f(h_t^{(l-1)}, h_{t-1}^{(l)}) + h_t^{(l-1)}$$

Gradient Flow:

$$\frac{\partial h_t^{(l)}}{\partial h_t^{(l-1)}} = \frac{\partial f}{\partial h_t^{(l-1)}} + I$$

The identity term I provides direct gradient flow, preventing vanishing even if $\frac{\partial f}{\partial h_t^{(l-1)}}$ is small.

Benefits:

- Direct gradient pathways across layers
- Improved training stability
- Better information flow in deep architectures

Question 6. Language Modeling Evaluation and Perplexity (18 marks)
Based on NLP evaluation metrics taught in university courses.

(a) Define perplexity as a measure of language model quality. Given a test sequence $w_1, w_2, \ldots, w_N$, derive the relationship between perplexity and cross-entropy loss. (8 marks)

Answer: Perplexity measures how well a language model predicts a text sequence, mathematically defined as the exponential of average cross-entropy.

Perplexity Definition: Perplexity measures the uncertainty or "surprise" of a language model when predicting text. Lower perplexity indicates better model performance.

Mathematical Derivation:

1. Probability of Sequence:

$$P(w_1, w_2, \ldots, w_N) = \prod_{i=1}^{N} P(w_i | w_1, \ldots, w_{i-1})$$

2. Log-likelihood:

$$\log P(w_1, \ldots, w_N) = \sum_{i=1}^{N} \log P(w_i | w_1, \ldots, w_{i-1})$$

3. Cross-entropy Loss:

$$H = -\frac{1}{N} \sum_{i=1}^{N} \log P(w_i | w_1, \ldots, w_{i-1})$$

4. Perplexity:

$$\text{PPL} = 2^H = 2^{-\frac{1}{N} \sum_{i=1}^{N} \log_2 P(w_i | w_1, \ldots, w_{i-1})}$$

Or equivalently: $\text{PPL} = \left(\prod_{i=1}^{N} P(w_i | w_1, \ldots, w_{i-1}) \right)^{-1/N}$

Relationship:

$$\text{Perplexity} = 2^{\text{Cross-entropy}}$$

Intuition: Perplexity represents the effective vocabulary size - a perplexity of 100 means the model is as confused as if it had to choose uniformly among 100 words at each step.

(b) A character-level RNN language model with vocabulary size 50 achieves the following results: (10 marks)

- Training perplexity: 1.8
- Validation perplexity: 2.3
- Test perplexity: 2.5

- Calculate the average bits per character for each dataset
- Analyze what these results indicate about model performance
- Compare with a baseline uniform model (calculate its perplexity)
- Suggest improvements to reduce the validation-test gap

Answer: Bits per character calculated using log of perplexity; results show good training but some overfitting compared to uniform baseline.

1. Bits Per Character Calculation: Since PPL $= 2^{\text{Cross-entropy}}$, we have Cross-entropy $= \log_2(\text{PPL})$:

- Training: $\log_2(1.8) = 0.85$ bits/char
- Validation: $\log_2(2.3) = 1.20$ bits/char
- Test: $\log_2(2.5) = 1.32$ bits/char

2. Performance Analysis:

- **Good Training Performance:** PPL of 1.8 is quite good for character-level modeling
- **Moderate Overfitting:** Validation PPL (2.3) $>$ Training PPL (1.8), indicating some overfitting
- **Generalization Gap:** Test PPL (2.5) slightly higher than validation, suggesting model generalizes reasonably well
- **Overall Assessment:** Model has learned meaningful patterns but could benefit from regularization

3. Uniform Baseline: For uniform distribution over 50 characters:

$$P_{\text{uniform}}(c) = \frac{1}{50} = 0.02$$

$$\text{PPL}_{\text{uniform}} = 50$$

$$\text{Bits per char}_{\text{uniform}} = \log_2(50) = 5.64 \text{ bits/char}$$

Our model (1.32 bits/char) significantly outperforms uniform baseline (5.64 bits/char).

4. Suggested Improvements:

- **Regularization:** Add dropout, weight decay, or early stopping
- **Data Augmentation:** Increase training data diversity
- **Architecture:** Try LSTM/GRU or deeper networks
- **Hyperparameter Tuning:** Optimize learning rate, hidden size
- **Ensemble Methods:** Combine multiple models
- **Better Preprocessing:** Improved tokenization or normalization

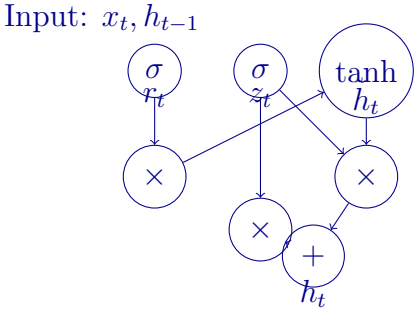
Question 7. Modern RNN Variants and Applications (27 marks)
Based on advanced topics from recent university deep learning curricula.

(a) Compare GRU (Gated Recurrent Unit) with LSTM architecture. Draw both architectures and explain: (12 marks)

- Key architectural differences
- Parameter count comparison
- Computational efficiency analysis
- When to choose GRU vs LSTM

Answer: GRU simplifies LSTM by combining forget and input gates into update gate and merging cell/hidden states.

GRU Architecture:



GRU Equations:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \quad \text{(reset gate)} \tag{14}$$
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \quad \text{(update gate)} \tag{15}$$
$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t]) \quad \text{(candidate)} \tag{16}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad \text{(final state)} \tag{17}$$

Key Architectural Differences:

- **States:** LSTM has separate cell/hidden states; GRU combines them
- **Gates:** LSTM has 3 gates (forget, input, output); GRU has 2 (reset, update)

- **Memory:** LSTM explicit memory cell; GRU implicit in hidden state
- **Control:** GRU update gate controls both forget and input simultaneously

Parameter Count Comparison: For hidden size h and input size d:

- **LSTM:** $4 \times (h \times (h + d) + h) = 4h(h + d + 1)$ parameters
- **GRU:** $3 \times (h \times (h + d) + h) = 3h(h + d + 1)$ parameters
- **Reduction:** GRU has 25% fewer parameters

Computational Efficiency:

- **GRU:** Fewer matrix multiplications, faster training/inference
- **LSTM:** More complex gating, higher memory requirements
- **Practical difference:** 10-15% speedup with GRU

When to Choose:

- **Choose GRU:** Limited data, computational constraints, simpler sequences
- **Choose LSTM:** Complex long-term dependencies, sufficient data, need explicit memory control

(b) Design a bidirectional RNN for named entity recognition. Explain: (8 marks)

- Forward and backward pass computations
- How to combine directional information
- Advantages for sequence labeling tasks
- Training considerations

Answer: Bidirectional RNN processes sequence in both directions, combining forward and backward hidden states for each position.

Architecture for NER:

1. Forward Pass:

$$\overrightarrow{h}_t = f(\overrightarrow{h}_{t-1}, x_t), \quad t = 1, 2, \ldots, T$$

2. Backward Pass:

$$\overleftarrow{h}_t = f(\overleftarrow{h}_{t+1}, x_t), \quad t = T, T-1, \ldots, 1$$

3. Combine Directional Information: Several approaches:

- **Concatenation:** $h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$
- **Sum:** $h_t = \overrightarrow{h}_t + \overleftarrow{h}_t$
- **Element-wise Product:** $h_t = \overrightarrow{h}_t \odot \overleftarrow{h}_t$
- **Learned Combination:** $h_t = W_f \overrightarrow{h}_t + W_b \overleftarrow{h}_t$

4. NER Prediction:

$$P(\text{label}_t) = \text{softmax}(W_o h_t + b_o)$$

Advantages for Sequence Labeling:

- **Full Context:** Access to both past and future context
- **Better Boundaries:** Can identify entity boundaries more accurately
- **Disambiguation:** Context from both directions helps resolve ambiguity
- **Example:** "Bank of America" - forward sees "Bank", backward sees "America"

Training Considerations:

- **Memory:** Double the memory requirement for hidden states
- **Parallelization:** Forward and backward passes can be computed in parallel
- **Gradient Flow:** Gradients flow through both directions
- **Loss Function:** Standard cross-entropy with BIO/BILOU tagging scheme

(c) Analyze the computational complexity of different RNN variants: (7 marks)

Model	Time Complexity	Space Complexity	Parameters
Vanilla RNN	?	?	?
LSTM	?	?	?
GRU	?	?	?
Bidirectional LSTM	?	?	?

For sequence length T, hidden size H, and input size D.

Answer: Complexity analysis shows linear scaling with sequence length, with LSTM/GRU having higher constants due to gating operations.

Complexity Analysis:

Model	Time Complexity	Space Complexity	Parameters
Vanilla RNN	$O(T \cdot H^2 + T \cdot H \cdot D)$	$O(T \cdot H)$	$O(H^2 + H \cdot D)$
LSTM	$O(T \cdot 4H^2 + T \cdot 4H \cdot D)$	$O(T \cdot 2H)$	$O(4H^2 + 4H \cdot D)$
GRU	$O(T \cdot 3H^2 + T \cdot 3H \cdot D)$	$O(T \cdot H)$	$O(3H^2 + 3H \cdot D)$
Bidirectional LSTM	$O(T \cdot 8H^2 + T \cdot 8H \cdot D)$	$O(T \cdot 4H)$	$O(8H^2 + 8H \cdot D)$

Detailed Breakdown:

Time Complexity Components:

- **Hidden-to-hidden:** $H \times H$ matrix multiplication at each step
- **Input-to-hidden:** $D \times H$ matrix multiplication at each step
- **Multiple gates:** Factor of 4 for LSTM, 3 for GRU
- **Sequence length:** Linear factor of T

Space Complexity:

- **Hidden states:** Store H values per time step
- **LSTM cell states:** Additional H values per time step
- **Bidirectional:** Double the space for both directions

Parameter Scaling:

- Parameters scale with H^2 (dominant term for large H)

- Input contribution scales with $H \cdot D$
- Gate multipliers: 4 for LSTM, 3 for GRU

Practical Implications:

- LSTM/GRU are 3-4x more expensive than vanilla RNN
- Bidirectional models double the computational cost
- Memory requirements can be significant for long sequences

END OF PAPER