Week 4: Sequence-to-Sequence Models

Post-Class Learning Verification

From Fixed-Length Prison to Variable-Length Freedom

INSTRUCTOR VERSION WITH ANSWER KEY

NLP Course 2025 - Assessment

Time Required: 45-60 minutes
Purpose: Verify and deepen your understanding of seq2seq models after completing the lab
Format: Conceptual problems, implementation questions, and real-world applications

Teaching Note

This assessment verifies student understanding after completing the seq2seq lab. Focus on conceptual understanding rather than mathematical precision. Common areas where students struggle: information bottleneck concept, attention mechanism intuition, and beam search vs greedy trade-offs.

Checkpoint

Before Starting: You should have completed the Week 4 lab and understood:

- Why variable-length sequences need special handling
- The encoder-decoder architecture
- The information bottleneck problem
- How attention mechanism works
- Beam search for sequence generation

Part A: Conceptual Understanding (25 minutes)

Teaching Note: Part A tests foundational concepts. Look for evidence students understand the core problem and solution.

A1: The Variable-Length Problem (5 minutes)

Understanding the Core Challenge

Question 1: Explain why traditional RNNs cannot handle translation tasks effectively.
Traditional RNNs produce exactly one output for each input time step, creating a fixed 1:1 mapping. Translation requires variable-length output (different languages use different numbers of words for the same concept). RNNs also suffer from the "final state compression" problem where all input information must be squeezed into a single final hidden state.

Question 2: Consider these translation pairs. Circle the fundamental problem:

- English: "How are you?" (3 words) → Spanish: "¿Cómo estás?" (2 words)
- English: "Thank you" (2 words) → German: "Vielen Dank" (2 words)
- English: "Good morning" (2 words) → Japanese: "Ohayou gozaimasu" (2 words)

✓ Different word counts between languages

✓ RNNs can only produce one output per input

☐ No shared vocabulary between languages *(This is not the core architectural problem)*

☐ Grammar structures are too different *(This is a challenge but not the fundamental limitation)*

Teaching Note: Students often focus on linguistic differences rather than the architectural limitation. Emphasize that the core issue is the rigid input-output mapping.

Question 3: List three real-world applications where variable-length input/output is essential:

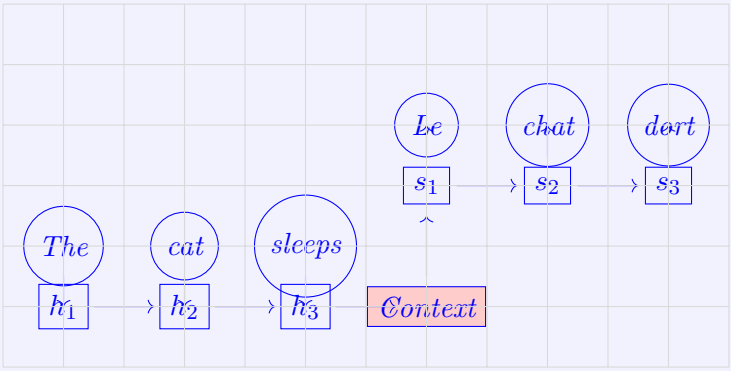
1. *Machine translation (different language structures)*
2. *Text summarization (long document → short summary)*
3. *Code generation (comment → complete function)*

Teaching Note: Accept various answers: chatbots, email auto-reply, question answering, speech-to-text, etc.

A2: Encoder-Decoder Architecture (8 minutes)

Understanding the Solution

Question 1: Draw and label the encoder-decoder architecture for translating "The cat sleeps" → "Le chat dort":



Sample answer shown in blue

Teaching Note: Accept any reasonable architectural diagram. Key elements: encoder RNNs, context vector bottleneck, decoder RNNs, variable-length output. Students often forget the context vector or don't show the information flow clearly.

Question 2: Complete the mathematical description:

- Encoder equations: $h_t =$ *LSTM(h_{t-1}, x_t) or RNN(h_{t-1}, x_t)*
- Context vector: $c =$ *h_T (final encoder state) or some function of all encoder states*
- Decoder equations: $s_t =$ *LSTM(s_{t-1}, y_{t-1}, c) - context influences decoder*
- Output probability: $P(y_t|\ldots) =$ *softmax($W_s s_t + b$) or similar linear transformation*

Teaching Note: Don't require exact notation. Look for understanding that encoder processes input sequentially, context captures information, decoder generates output conditioned on context.

Question 3: Why is the context vector a "bottleneck"?

The context vector has fixed size (e.g., 256 dimensions) regardless of input length. All information from a variable-length input sequence must be compressed into this fixed-size vector, causing information loss for long sequences. This is why translation quality degrades with sentence length in vanilla seq2seq models.

Question 4: If your encoder has 128 hidden units and processes a 20-word sentence, what is the size of:

- Context vector: *128* dimensions
- Total encoder information: *2560 (20 × 128)* dimensions
- Information compression ratio: *20:1 or 95% compression*

Teaching Note: This calculation helps students understand the severity of the bottleneck problem. 95% information loss is dramatic!

A3: Attention Mechanism Deep Dive (8 minutes)

Attention Mathematics and Intuition

Question 1: Complete the attention mechanism steps:
Step 1: Compute alignment scores: $e_{t,i} =$ *align(s_{t-1}, h_i) - similarity between decoder state and encoder state*
Step 2: Normalize with softmax: $\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{T} \exp(e_{t,j})}$ *- attention weights sum to 1*
Step 3: Compute context vector: $c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i$ *- weighted combination of all encoder states*
Question 2: Given these simplified encoder states and decoder state:

- $h_1 = [0.5, 0.2]$ (word: "The")
- $h_2 = [0.8, 0.1]$ (word: "cat")
- $h_3 = [0.3, 0.7]$ (word: "sleeps")
- $s_t = [0.7, 0.3]$ (generating French word)

Calculate dot-product attention weights:

- $e_{t,1} = h_1 \cdot s_t =$ *0.5×0.7 + 0.2×0.3 = 0.35 + 0.06 = 0.41*
- $e_{t,2} = h_2 \cdot s_t =$ *0.8×0.7 + 0.1×0.3 = 0.56 + 0.03 = 0.59*
- $e_{t,3} = h_3 \cdot s_t =$ *0.3×0.7 + 0.7×0.3 = 0.21 + 0.21 = 0.42*

Which word gets highest attention? *"cat" (h_2)withscore*0.59
Teaching Note: *Walk through the calculation step by step. After softmax, the weights would be approximately [0.27, 0.42, 0.31], so "cat" gets highest attention.*
Question 3: *Attention visualization interpretation. If you see this attention pattern:*

French	***The***	***black***	***cat***	***sleeps***
Le	*0.8*	*0.1*	*0.1*	*0.0*
chat	*0.1*	*0.1*	*0.8*	*0.0*
noir	*0.0*	*0.9*	*0.1*	*0.0*
dort	*0.0*	*0.0*	*0.1*	*0.9*

What pattern do you observe? Nearly diagonal alignment - each French word attends strongly to its corresponding English word. This shows the model learned word-to-word correspondence.
Why is this good for translation? It shows the model is learning meaningful alignments between source and target languages, rather than just using averaged information. Each output word can focus on the most relevant input words.
Teaching Note: *This is an idealized attention pattern. Real patterns are usually noisier but should show some meaningful structure.*

A4: Beam Search Strategy (4 minutes)

Search and Generation

Question 1: Why can't we just use greedy search (always pick highest probability word)?
Greedy search can get stuck in locally optimal but globally suboptimal paths. Early high-probability words might lead to poor overall sequences. For example, starting with "The" might be locally optimal but "A" could lead to a better overall translation. Beam search explores multiple promising paths simultaneously.
Question 2: Complete this beam search example (beam size = 2):
Starting with "START", expand to first word:

- P("Le" — START) = 0.6
- P("Un" — START) = 0.3
- P("La" — START) = 0.1

Keep top 2: *"Le"* and *"Un"*
Expand "Le" to second word:

- P("chat" — Le) = 0.8 → Total: $0.6 \times 0.8 = 0.48$
- P("chien" — Le) = 0.2 → Total: $0.6 \times 0.2 = 0.12$

Expand "Un" to second word:

- P("chat" — Un) = 0.7 → Total: $0.3 \times 0.7 = 0.21$
- P("chien" — Un) = 0.3 → Total: $0.3 \times 0.3 = 0.09$

Final top 2 sequences: *"Le chat" (0.48)* and *"Un chat" (0.21)*
Teaching Note: Make sure students understand that we keep the globally best paths, not just the best extensions of each current path.
Question 3: What happens to beam search quality vs. beam size?
Beam size 1: *Equivalent to greedy search; fast but potentially poor quality*
Beam size 100: *High quality but computationally expensive; may overfit to training patterns*
Optimal beam size (typical): *4-8 for translation tasks; balances quality and speed*
Teaching Note: Emphasize the trade-off. In practice, beam size depends on the application and computational constraints.

Part B: Implementation and Code Understanding (15 minutes)

Teaching Note: Part B tests practical understanding. Students should be able to read and debug code even if they can't write it from scratch.

B1: Code Analysis (8 minutes)

PyTorch Implementation

Question 1: Analyze this encoder code. Fill in the missing dimensions:

```
class Seq2SeqEncoder(nn.Module):
    def __init__(self, vocab_size=5000, embed_size=128, hidden_size=256):
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, batch_first=True)

    def forward(self, x):  # x shape: [batch_size, seq_len]
        embedded = self.embedding(x)  # shape: [______, ______, ______]
        output, (h_n, c_n) = self.lstm(embedded)
        return h_n, c_n  # shapes: [______, ______], [______, ______]
```

Fill in the shapes:

- embedded shape: [*batch_size*, *seq_len*, *128*]
- h_n shape: [*1*, *256*] *(assuming 1 layer, batch_size implicit)*
- c_n shape: [*1*, *256*] *(same as h_n for LSTM)*

Teaching Note: Shape understanding is crucial for debugging. Students often forget that LSTM returns states with layer dimension first.

Question 2: What's wrong with this attention implementation?

```
def attention(decoder_hidden, encoder_outputs):
    scores = torch.matmul(decoder_hidden, encoder_outputs.T)
    weights = F.softmax(scores, dim=1)
    context = torch.sum(weights * encoder_outputs, dim=1)
    return context
```

Problem: *Dimension mismatch. decoder_hidden is likely [batch, hidden] and encoder_outputs is [batch, seq_len, hidden]. The transpose and matrix multiplication don't align properly.*

Fix: *Use torch.bmm for batch matrix multiplication or expand decoder_hidden to match encoder_outputs dimensions, or use Einstein summation: torch.einsum('bh,bsh-¿bs', decoder_hidden, encoder_outputs)*

Question 3: Complete this beam search pseudocode:

```
def beam_search(model, input_seq, beam_size=4, max_length=20):
    beams = [{"sequence": [START_TOKEN], "score": 0.0}]

    for step in range(max_length):
        candidates = []
        for beam in beams:
            if beam["sequence"][-1] == END_TOKEN:
                candidates.append(beam)
                continue

            # Get next word probabilities
            probs = model.predict_next(beam["sequence"])

            # Expand beam
            for word, prob in probs.top_k(beam_size):
                new_score = ____________________
                new_sequence = ______________
```

B2: Training Insights (7 minutes)

Training Process

Question 1: What is "teacher forcing" and why do we use it during training?
Definition: *During training, use the true target tokens as decoder inputs instead of the model's own predictions. Feed the correct previous word even if the model predicted something else.*
Why use it: *Accelerates training by providing stable gradients and prevents error accumulation during training. Allows parallel computation across the target sequence.*
What problem does it cause: *Exposure bias - at inference time, the model must use its own predictions, creating a mismatch between training and testing conditions. Can lead to error accumulation during generation.*
Teaching Note: This is a subtle but important concept. Students often struggle with understanding why training differs from inference in seq2seq models.
Question 2: Loss function analysis. Given these target and predicted sequences:
Target: ["Le", "chat", "dort", "¡EOS¿"] Predicted logits for each position:

- Position 1: P("Le")=0.8, P("Un")=0.15, P("La")=0.05
- Position 2: P("chat")=0.9, P("chien")=0.07, P("oiseau")=0.03
- Position 3: P("dort")=0.6, P("mange")=0.3, P("court")=0.1
- Position 4: P("¡EOS¿")=0.95, P("bien")=0.03, P("mal")=0.02

Calculate the cross-entropy loss:

- Position 1 loss: $-\log(0.8) =$ *0.097*
- Position 2 loss: $-\log(0.9) =$ *0.046*
- Position 3 loss: $-\log(0.6) =$ *0.222*
- Position 4 loss: $-\log(0.95) =$ *0.022*
- Total loss: *0.387 (sum of individual losses)*

Teaching Note: Don't require precise calculations. Look for understanding that lower probabilities give higher losses, and total loss is the sum.
Question 3: Why might attention weights look random at the start of training?
At initialization, the alignment function hasn't learned meaningful relationships yet. The similarity scores between decoder and encoder states are essentially random, leading to uniform or noisy attention distributions. The model hasn't learned which source words are relevant for predicting specific target words.
What should happen to attention weights as training progresses?
Attention should become more focused and structured. For translation, we expect somewhat diagonal patterns (source word i attends to target word j). The model learns which source positions are most relevant for each target position, leading to sharper, more interpretable attention patterns.

Part C: Real-World Applications (12 minutes)

Teaching Note: Part C connects concepts to practical applications. Look for understanding of how seq2seq principles apply to modern systems.

C1: Modern Applications Analysis (6 minutes)

2024 Industry Applications

Question 1: Map these modern systems to seq2seq components:

Application	Input	Output
Google Translate	*Source language text*	*Target language text*
GitHub Copilot	*Code context/comments*	*Generated code*
Email Summarization	*Full email content*	*Short summary*
Text-to-SQL	*Natural language query*	*SQL statement*
Chatbot Response	*User message/context*	*Bot response*

Question 2: Which of these would benefit most from attention mechanism?

- ☐ Translating short phrases (3-5 words) *Less critical for short sequences*
- ✓ Translating technical documents (500+ words) *Most benefit - long sequences suffer from bottleneck*
- ☐ Generating code from one-line comments *Moderate benefit*
- ✓ Summarizing research papers *High benefit - long input, selective attention needed*
- ☐ Converting speech to text *Different domain, but attention helps with alignment*

Explain your top choice: *Long documents suffer most from information bottleneck. Attention allows the model to focus on different parts of the document when generating different parts of the summary, rather than compressing everything into a single context vector.*

Question 3: Compare 2014 vs 2024 seq2seq capabilities:

Aspect	2014 (Original)	2024 (Modern)
Max input length	*20 words practical*	*1000s of tokens*
Translation quality	*25-30 BLEU*	*65+ BLEU*
Inference speed	*Slow (sequential)*	*Fast (parallel/optimized)*
Model size	*10M parameters*	*100B+ parameters*
Applications	*Research demos*	*Production systems*

Teaching Note: Accept reasonable estimates. The key is understanding the massive scaling in all dimensions.

C2: System Design Challenge (6 minutes)

Building Real Systems

Question 1: Design a meeting summarization system:
Input: 2-hour meeting transcript (5000 words) **Output:** Key points and action items (200 words)
Your architecture:

- Preprocessing: *Segment into sentences, remove filler words, identify speakers, timestamp key sections*
- Encoder design: *Hierarchical encoder - sentence-level then document-level, or transformer with positional encoding for long sequences*
- Attention strategy: *Multi-head attention to focus on different types of content (decisions, action items, key points)*
- Decoder design: *Structured decoder that generates specific sections (decisions, actions, participants) with appropriate formatting*
- Post-processing: *Format into bullet points, add timestamps, validate action item assignments*

Question 2: Code comment to function generator:
Input: "Function to calculate fibonacci numbers up to n" **Output:** Complete Python function
What challenges would this face?

1. *Ambiguous specifications - many ways to implement fibonacci*
2. *Variable function signatures - what parameters, return types?*
3. *Style and convention consistency - naming, formatting, documentation*

How would attention help? *Attention allows the model to focus on different parts of the comment when generating different parts of the code. For example, attending to "fibonacci" when deciding on algorithm, "up to n" when handling the parameter, etc.*
Question 3: Multilingual customer support system:
Requirements:

- Input: Customer query in any of 10 languages
- Output: Response in same language as input
- Must handle domain-specific terminology

Design approach:

- Language detection: *Classifier to identify input language before processing*
- Translation pipeline: *Translate to English, process, translate back; or use multilingual model*
- Response generation: *Seq2seq model trained on customer service conversations with domain-specific vocabulary*
- Quality assurance: *Confidence scoring, human review for low-confidence responses, feedback loop for continuous improvement*

Teaching Note: Look for practical considerations: language detection, domain adaptation, quality control, human-in-the-loop systems.

Part D: Critical Thinking and Extensions (8 minutes)

D1: Problem Solving (4 minutes)

Common Pitfall

Real challenges you might encounter in production:

Debugging and Optimization

Scenario 1: Your seq2seq translator produces repetitive output: "The cat the cat the cat sleeps"
Possible causes:

- ☐ Attention weights are too uniform *Possible but not the main cause*
- ✓ Beam search beam size too small *Yes - can get stuck in repetitive loops*
- ✓ Training data has repetitive patterns *Yes - model learns from data artifacts*
- ☐ Learning rate too high *Less likely to cause this specific pattern*
- ✓ Decoder getting stuck in local optima *Yes - especially with greedy search*

Best solution: *Use coverage mechanisms, increase beam diversity, add repetition penalties during decoding, or train with more diverse data. Also check for exposure bias from teacher forcing.*
Scenario 2: Translation quality drops drastically for sentences longer than 20 words.
Root cause: *Information bottleneck problem - fixed-size context vector cannot capture all information from long sequences. Earlier parts of the sequence are "forgotten" by the time encoding is complete.*
Solution strategy: *Implement attention mechanism to allow decoder to access all encoder states, not just the final one. This eliminates the fixed-size bottleneck and allows the model to selectively focus on relevant parts of the input.*
Scenario 3: Your model works great on formal text but fails on casual social media language.
Why this happens: *Domain mismatch between training data (likely formal text) and test data (casual language). Different vocabulary, grammar, abbreviations, and informal expressions not seen during training.*
How to fix: *Domain adaptation: fine-tune on social media data, augment training with informal text, use domain adversarial training, or preprocess to normalize informal language to formal equivalents.*
Teaching Note: These scenarios reflect real production challenges. Emphasize that data and domain considerations are as important as model architecture.

D2: Future Connections (4 minutes)

Think About It

Connecting to Advanced Topics:
Question 1: How do Transformers (next week) improve on seq2seq?
Key improvements:

1. *Parallelization - self-attention allows parallel processing vs sequential RNN computation*
2. *Better long-range dependencies - attention connects distant positions directly*
3. *Multiple attention heads - can attend to different types of relationships simultaneously*

Question 2: Modern large language models (ChatGPT, GPT-4) use modified seq2seq principles. What's different?
Architectural changes: *Transformer-based (self-attention only), decoder-only architectures for autoregressive generation, much deeper networks with residual connections*
Scale differences: *Billions of parameters vs millions, trained on trillions of tokens vs millions, massive computational resources*
Training approach: *Pre-training on massive general text + fine-tuning/RLHF, rather than task-specific training from scratch*
Question 3: Beyond text, what other sequence-to-sequence problems exist?

- Audio domain: *Speech-to-text, music generation, audio translation*
- Video domain: *Video captioning, action recognition, video prediction*
- Scientific domain: *Protein folding prediction, chemical reaction prediction, DNA sequence analysis*
- Creative domain: *Image captioning, story generation, code documentation*

Teaching Note: Help students see that seq2seq is a general framework beyond just translation. Many AI problems can be framed as sequence-to-sequence tasks.

Grading Rubric

Teaching Note

Point Distribution (Total: 100 points)

- Part A: 40 points (Conceptual understanding - most critical)
- Part B: 25 points (Implementation understanding)
- Part C: 25 points (Real-world applications)
- Part D: 10 points (Critical thinking and connections)

Grading Guidelines:

- Full credit: Demonstrates complete understanding with correct explanations
- Partial credit (75%): Mostly correct with minor gaps or inaccuracies
- Half credit (50%): Shows understanding but significant errors or omissions
- Minimal credit (25%): Attempted but fundamental misunderstandings
- No credit (0%): No attempt or completely incorrect

Common Student Errors:

- Confusing encoder and decoder roles
- Not understanding the bottleneck problem severity
- Thinking attention "replaces" the context vector rather than accessing all encoder states
- Confusing teacher forcing with beam search
- Focusing on linguistic differences rather than architectural limitations

Discussion Points for Class Review:

- Why was seq2seq such a breakthrough? (Variable length capability)
- How does attention solve the bottleneck? (Access to all encoder states)
- Connection to modern transformers (self-attention evolution)
- Real-world considerations (data quality, domain adaptation, computational costs)

Self-Assessment and Next Steps

Checkpoint

Check your understanding level:

- ☐ I can explain why seq2seq was needed (vs fixed-length RNNs)
- ☐ I understand the encoder-decoder architecture completely
- ☐ I can describe the information bottleneck problem
- ☐ I can explain how attention solves the bottleneck
- ☐ I can implement attention mechanism from scratch
- ☐ I understand beam search vs greedy search trade-offs
- ☐ I can design seq2seq systems for real applications
- ☐ I see the connection to modern transformer models

Teaching Note: Students checking fewer than 6 items need additional review. Consider office hours or peer tutoring.

Real World Application

Industry Relevance Check: Rate your confidence (1-5) for these career-relevant skills:

- Building translation systems: *Target: 3-4 for BSc students*
- Designing text summarization: *Target: 3-4 for BSc students*
- Code generation tools: *Target: 2-3 for BSc students*
- Conversational AI systems: *Target: 3-4 for BSc students*
- Understanding modern LLM architecture: *Target: 4-5 for BSc students*

Areas needing review: *Most commonly: attention mechanism details, beam search vs greedy trade-offs, connection to transformers*

Most interesting discovery: *Often: how attention solves the bottleneck problem, or connection to modern AI systems they use daily*

Connection to your projects/interests: *Look for genuine engagement - these students may benefit from research projects or advanced courses*

Next Steps

Immediate review: Focus on areas where you scored below 3/5

Hands-on practice:

- Implement seq2seq from scratch in your preferred framework
- Try the model on a different language pair
- Experiment with different attention mechanisms

- Build a simple summarization system

Preparation for Week 5:

- Review attention mechanism thoroughly
- Understand the limitations of RNN-based seq2seq
- Think about parallelization challenges
- Read "Attention Is All You Need" paper introduction

Teaching Note: Encourage students to implement a small seq2seq project before moving to transformers. The hands-on experience helps solidify concepts.

— End of Assessment —

You're now ready to understand how Transformers revolutionized this field!