Week 4 Quick Reference: Sequence-to-Sequence Models

Essential Concepts, Equations, and Code Patterns

NLP Course 2025

Core Problem & Solution

Variable-Length Challenge

Problem: Traditional RNNs require fixed input-output mapping

- Each input → exactly one output
- Translation needs variable lengths
- Summarization: long → short
- Code generation: comment → function

Encoder-Decoder Solution

Innovation: Separate encoding from decoding

- **Encoder:** Input sequence → context vector
- **Decoder:** Context vector → output sequence
- **Result:** Variable input/output lengths

Key Equations

Basic Seq2Seq

Encoder: $h_t = \text{LSTM}(h_{t-1}, x_t)$

$c = h_T$ (context vector)

Decoder: $s_t = \text{LSTM}(s_{t-1}, y_{t-1}, c)$

$P(y_t|y_{<t}, x) = \text{softmax}(W_s s_t + b)$

Attention Mechanism

Alignment Score: $e_{t,i} = \text{align}(s_{t-1}, h_i)$

Attention Weights: $\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{T} \exp(e_{t,j})}$

```
attention_weights = F.softmax(attention_scores, dim=1)

context = torch.bmm(attention_weights.unsqueeze(1), encoder_outputs).
        squeeze(1)
return context, attention_weights
```

Context Vector: $c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i$

Attention Types

- **Dot Product:** $e_{t,i} = s_t \cdot h_i$
- **Scaled Dot:** $e_{t,i} = \frac{s_t \cdot h_i}{\sqrt{d}}$
- **Additive:** $e_{t,i} = v^T \tanh(W_s s_t + W_h h_i)$

PyTorch Implementation

Basic Encoder

```
class Encoder(nn.Module):
    def __init__(self, vocab_size, embed_size, hidden_size):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, batch_first=True)

    def forward(self, x):
        embedded = self.embedding(x)
        output, (h_n, c_n) = self.lstm(embedded)
        return output, h_n, c_n  # All states + final states
```

Attention Module

```
class Attention(nn.Module):
    def __init__(self, hidden_size):
        super().__init__()
        self.W_a = nn.Linear(hidden_size, hidden_size)
        self.U_a = nn.Linear(hidden_size, hidden_size)
        self.v_a = nn.Linear(hidden_size, 1)

    def forward(self, decoder_hidden, encoder_outputs):
        # decoder_hidden: [batch, hidden_size]
        # encoder_outputs: [batch, seq_len, hidden_size]
        seq_len = encoder_outputs.size(1)
        decoder_hidden = decoder_hidden.unsqueeze(1).repeat(1, seq_len, 1)

        energy = torch.tanh(self.W_a(decoder_hidden) + self.U_a(
            encoder_outputs))
        attention_scores = self.v_a(energy).squeeze(2)
```

Beam Search

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

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

            probs = model.predict_next(beam['seq'])
            for word, prob in probs.top_k(beam_size):
                new_score = beam['score'] + log(prob)
                new_seq = beam['seq'] + [word]
                candidates.append({'seq': new_seq, 'score': new_score})

        beams = sorted(candidates, key=lambda x: x['score'])[-beam_size:]

    return beams[0]['seq']
```

Key Concepts

Information Bottleneck

Problem: Single context vector can't hold all information

- Fixed size (e.g., 256 dimensions)
- Variable input length
- Information loss for long sequences
- Quality degrades with length

Solution: Attention mechanism

Teacher Forcing

Training: Use true target tokens as decoder input

- Faster convergence
- Stable gradients
- Parallel computation possible

Problem: Exposure bias (training inference)

Beam Search vs Greedy

Method	Quality	Speed
Greedy	Low	Fast
Beam (k=4)	High	Medium
Beam (k=20)	Highest	Slow

Common Issues & Solutions

Repetitive Output

Symptoms: "The cat the cat the cat"

- **Cause:** Beam search stuck in loops
- **Fix:** Add repetition penalty
- **Fix:** Increase beam diversity
- **Fix:** Coverage mechanisms

Poor Long Sequences

Symptoms: Quality drops after 20+ words

- **Cause:** Information bottleneck
- **Fix:** Add attention mechanism
- **Fix:** Increase context vector size
- **Prevention:** Use attention from start

Training Issues

Symptoms: Model not learning

- **Check:** Teacher forcing implementation
- **Check:** Gradient clipping (RNN vanishing gradients)
- **Check:** Learning rate schedule
- **Check:** Sequence length distribution

Attention Variants

Type	Formula	Used In
Dot Product	$s_t \cdot h_i$	Simple models
Scaled Dot	$\frac{s_t \cdot h_i}{\sqrt{d_k}}$	Transformers
Additive	$v^T \tanh(W_s s_t + W_h h_i)$	Bahdanau (2015)
Multiplicative	$s_t^T W h_i$	Luong (2015)

Performance Metrics

BLEU Score

Translation Quality Metric:

- 0-100 scale (higher = better)
- Based on n-gram precision
- Industry standard for translation
- Typical scores:
  - Poor: 0-20
  - Acceptable: 20-40
  - Good: 40-60
  - Excellent: 60+

Historical Timeline

Year	Innovation	BLEU
2014	Vanilla Seq2Seq	25
2015	Attention Added	35
2016	Google NMT	45
2017	Transformer	52
2024	Modern LLMs	68+

Modern Applications (2024)

Translation Systems

- **Google Translate:** 1B+ daily translations
- **DeepL:** 1B people served monthly
- **Architecture:** Transformer encoder-decoder
- **Languages:** 100+ language pairs

Code Generation

- **GitHub Copilot:** 1M+ developers
- **Task:** Comment → code function
- **Productivity:** 40% faster development
- **Architecture:** GPT-based encoder-decoder

Text Summarization

- **Email Apps:** Outlook, Gmail auto-summaries
- **News:** Automatic headline generation
- **Research:** Paper → abstract
- **Architecture:** Encoder-decoder with attention

Conversational AI

- **ChatGPT:** 100M+ users
- **Task:** Context → response
- **Architecture:** Transformer decoder (GPT)
- **Innovation:** Self-attention evolution

Implementation Checklist

Building Encoder

- ☐ Embedding layer for input tokens
- ☐ LSTM/GRU for sequence processing
- ☐ Return all hidden states (for attention)
- ☐ Return final state (for decoder init)

Building Decoder

- ☐ Embedding layer for output tokens
- ☐ LSTM/GRU initialized with encoder state
- ☐ Linear layer for vocabulary projection
- ☐ Softmax for probability distribution

Adding Attention

- ☐ Alignment function (dot product/additive)
- ☐ Softmax normalization
- ☐ Weighted context computation
- ☐ Concatenate context with decoder input

Training Setup

- ☐ Teacher forcing during training
- ☐ Cross-entropy loss
- ☐ Gradient clipping (important for RNNs)
- ☐ Learning rate scheduling

Hyperparameters

Parameter	Typical Range	Notes
Hidden Size	128-512	Larger for complex tasks
Embed Size	128-300	Often = hidden size
Num Layers	1-4	Deeper = more capacity
Dropout	0.1-0.3	Prevent overfitting
Learning Rate	1e-4 to 1e-3	Adam optimizer
Beam Size	4-8	Quality vs speed trade-off
Max Length	50-200	Task dependent

Troubleshooting Guide

Model Not Learning

Check:

- Learning rate too high/low
- Gradient clipping threshold
- Teacher forcing ratio
- Sequence length handling

Poor Attention

Check:

- Attention weights visualization
- Alignment function choice
- Hidden size compatibility
- Training data quality

Bad Beam Search

Check:

- Beam size vs vocabulary size
- Length normalization
- Repetition penalty
- End token handling

Key Papers & Resources

Foundational Papers

- Sutskever et al. (2014): "Sequence to Sequence Learning with Neural Networks"
- Bahdanau et al. (2015): "Neural Machine Translation by Jointly Learning to Align and Translate"
- Luong et al. (2015): "Effective Approaches to Attention-based Neural Machine Translation"

Modern Context

- Wu et al. (2016): "Google's Neural Machine Translation System"
- Vaswani et al. (2017): "Attention Is All You Need" (Transformer)

Visualization Resources

- Jay Alammar: "Visualizing A Neural Machine Translation Model"
- The Illustrated Transformer
- Hugging Face Transformers documentation

Connection to Week 5: Transformers Preview

Next Week Preview: From Seq2Seq to "Attention Is All You Need"

What Transformers Keep:
- Encoder-decoder architecture
- Attention mechanism
- Variable-length I/O
- Teacher forcing training

What Transformers Change:
- Remove RNNs entirely
- Add self-attention
- Enable parallelization
- Multi-head attention

The Evolution: Seq2Seq (2014) → Attention (2015) → Transformer (2017) → GPT/BERT (2018+) → ChatGPT (2022) → Modern AI (2024)
Your Journey: Master seq2seq this week → Understand transformers next week → Comprehend modern AI architecture!

Quick Reference Complete — *Keep this handy during lab sessions!* — **NLP Course 2025**