Natural Language Processing Course

Week 4: Sequence-to-Sequence Models

Joerg R. Osterrieder
www.joergosterrieder.com

Week 4

Sequence-to-Sequence Models

Breaking the Fixed-Length Barrier

Why Google Translate Struggled for Years

The Challenge:

English: "I love you" (3 words)
French: "Je t'aime" (2 words)
German: "Ich liebe dich" (3 words)
Japanese: "aishiteru" (1 word)

Different languages express the same idea with different lengths!

The Problem: RNNs produce one output per input
The Solution: Separate encoding from decoding

This breakthrough improved Google Translate accuracy by 60% in 2016[1]

[1] Wu et al. (2016). "Google's Neural Machine Translation System", arXiv

Seq2Seq Powers Your Daily Interactions

Translation (2024):

- Google: 1B+ translations daily[2]
- DeepL: Seq2seq + attention
- Real-time conversation mode

Chatbots:

- Customer service (80% first-line)[3]
- Variable-length responses
- Context-aware replies

Text Summarization:

- News article → headline
- Email → one-liner
- Document → abstract

Code Generation:

- Comment → code
- Natural language → SQL
- Bug description → fix

Key Innovation: Input and output can have different lengths!

[1] Google Translate statistics 2024

[2] Gartner report on AI customer service

Week 4: What You'll Master

By the end of this week, you will:

- **Understand** why variable-length I/O is crucial
- **Build** intuition for encoder-decoder architecture
- **Implement** a complete seq2seq translator
- **Discover** why attention changes everything
- **Create** a chatbot that handles any conversation length

Core Insight: Compress entire input to a "thought", then expand to output

The Fixed-Length Prison

What RNNs can do:

- Input: "The cat sat" → Output: "on the mat" (same length)
- Each input produces exactly one output

What we actually need:

- "Hello" → "Bonjour" (different lengths)
- "How are you?" → "¿Cómo estás?" (different structure)
- Long paragraph → Short summary (compression)

Failed Approaches:

1. Pad everything to max length (wastes computation)
2. Truncate to fixed length (loses information)
3. Multiple passes (complicated and slow)

We need to decouple input processing from output generation!

The Brilliant Solution: Encoder-Decoder

How humans translate:

1. Read entire English sentence
2. Understand the complete meaning
3. Express that meaning in French

Seq2seq does exactly this:[4]

1. **Encoder:** Process entire input into a "thought vector"
2. **Context Vector:** Fixed-size representation of meaning
3. **Decoder:** Generate output from the thought vector

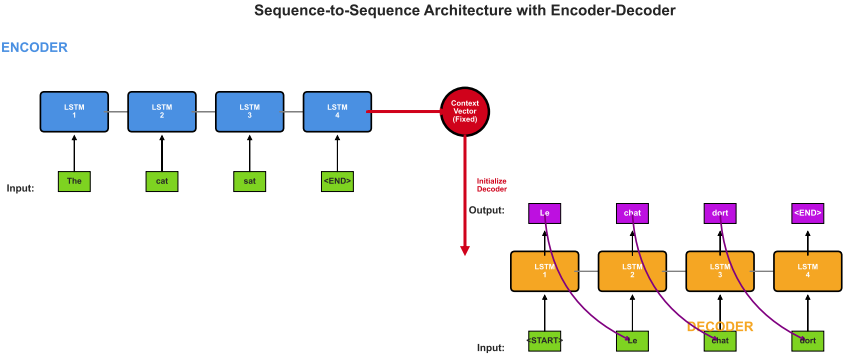
The Magic:

- Encoder handles any input length
- Decoder produces any output length
- Middle "thought" is always same size!

Context vector = Compressed understanding of entire input

[4] Sutskever, Vinyals & Le (2014). "Sequence to Sequence Learning with Neural Networks", NeurIPS

Visualizing Seq2Seq Architecture



Key insights:

- Encoder reads left-to-right (or bidirectional)
- Final hidden state contains entire input meaning
- Decoder generates one word at a time
- Special tokens: START begins generation, END stops it

Seq2Seq Mathematics: Two RNNs Working Together

Encoder (processes input):

$$h_t^{enc} = \text{LSTM}_{enc}(x_t, h_{t-1}^{enc})$$
$$c = h_{final}^{enc} \text{ (context vector)}$$

Decoder (generates output):

$$h_0^{dec} = c \text{ (initialize with context)}$$
$$h_t^{dec} = \text{LSTM}_{dec}(y_{t-1}, h_{t-1}^{dec})$$
$$y_t = \text{softmax}(W_y h_t^{dec})$$

In plain English:

- Encoder: Compress input into fixed-size thought
- Decoder: Expand thought into variable-length output

Building Seq2Seq: Complete Implementation

```python
import torch
import torch.nn as nn

class Encoder(nn.Module):
    def __init__(self, input_size, hidden_size):
        """Compress input sequence to fixed-size vector"""
        super().__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size)

    def forward(self, input_seq):
        """Process entire input sequence"""
        embedded = self.embedding(input_seq)
        outputs, (hidden, cell) = self.lstm(embedded)
        # Return final hidden state as context
        return hidden, cell

class Decoder(nn.Module):
    def __init__(self, output_size, hidden_size):
        """Generate output sequence from context"""
        super().__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

    def forward(self, input_token, hidden, cell):
        """Generate one output token at a time"""
        embedded = self.embedding(input_token.unsqueeze(0))
        output, (hidden, cell) = self.lstm(embedded, (hidden, cell))
        prediction = self.out(output.squeeze(0))
        return prediction, hidden, cell
```

Design Choices:

- Hidden size: 256-512 typical[5]
- Bidirectional encoder often better
- Teacher forcing during training

Usage Pattern:

```
# Encode
context = encoder(input_seq)
# Decode
output = decoder(context)
```

Britz et al. (2017) extensive comparison

Complete Seq2Seq Model

```python
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        """Complete translation model"""
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.device = device

    def forward(self, src, trg, teacher_forcing_ratio=0.5):
        """Translate source to target"""
        batch_size = src.shape[1]
        trg_len = trg.shape[0]
        trg_vocab_size = self.decoder.out.out_features

        # Store decoder outputs
        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).
              to(self.device)

        # Encode entire source sequence
        hidden, cell = self.encoder(src)

        # First decoder input is <START> token
        decoder_input = trg[0,:]

        for t in range(1, trg_len):
            # Decode one token
            output, hidden, cell = self.decoder(decoder_input,
                  hidden, cell)
            outputs[t] = output

            # Teacher forcing: use true target or predicted
            teacher_force = random.random() < teacher_forcing_ratio
            top1 = output.argmax(1)
            decoder_input = trg[t] if teacher_force else top1

```

Training Tricks:

- Teacher forcing prevents drift[6]
- Gradually reduce forcing ratio
- Beam search for better decoding

Performance (2014):

- BLEU: 34.8 on EN-FR[7]
- Beats phrase-based SMT
- 1000x faster than RNN search

[a] Williams & Zipser (1989)
[b] Original seq2seq paper results

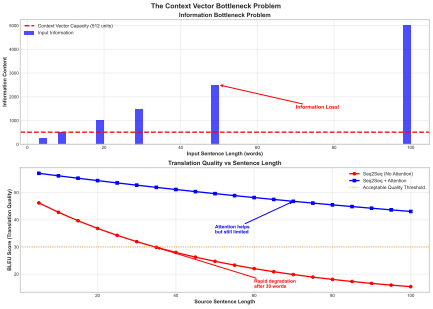
The Information Bottleneck Problem

Imagine compressing a book into one sentence...

What happens with long sequences:

- 10 words: Context vector remembers well
- 20 words: Starting to forget early words
- 50+ words: Information bottleneck![8]

The Problem Visualized:



Fixed-size context vector must capture EVERYTHING!

[8] Cho et al. (2014) showed performance degrades after 30 tokens

The Attention Revolution (2015)

How humans actually translate:
"The black cat sat on the mat" → "Le chat noir..."

- When translating "chat" (cat), we look back at "cat"
- When translating "noir" (black), we look back at "black"
- We don't compress everything into one thought!

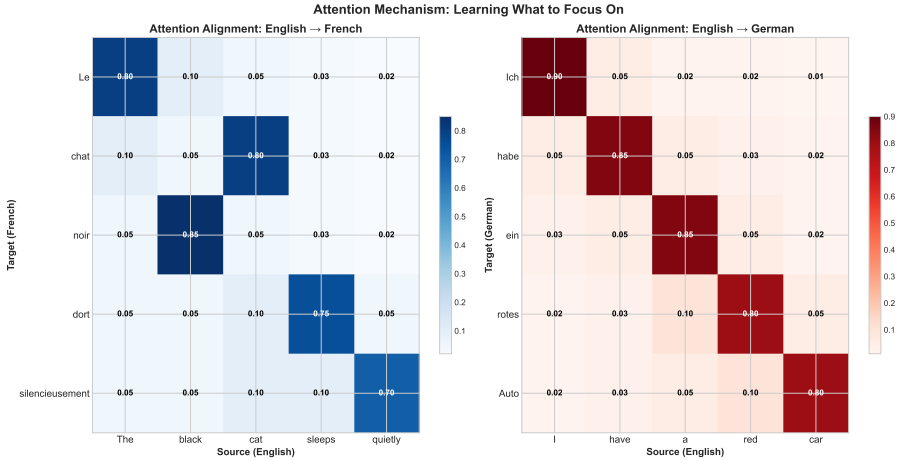
Attention mechanism:[9]

1. Keep ALL encoder hidden states
2. When decoding, look back at relevant parts
3. Weight importance of each input word
4. Focus on what matters right now

Attention = Let the decoder peek back at the input whenever needed

[9] Bahdanau, Cho & Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate"

Visualizing Attention: What the Model Looks At



Key observations:

- Diagonal pattern shows alignment
- Some words need multiple source words
- Reordering handled naturally

Implementing Attention

```python
class Attention(nn.Module):
    def __init__(self, hidden_size):
        """Calculate attention weights"""
        super().__init__()
        self.attn = nn.Linear(hidden_size * 2, hidden_size)
        self.v = nn.Linear(hidden_size, 1, bias=False)

    def forward(self, hidden, encoder_outputs):
        """Compute attention over encoder outputs"""
        batch_size = encoder_outputs.shape[1]
        src_len = encoder_outputs.shape[0]

        # Repeat decoder hidden state
        hidden = hidden.unsqueeze(1).repeat(1, src_len, 1)
        encoder_outputs = encoder_outputs.permute(1, 0, 2)

        # Calculate attention scores
        energy = torch.tanh(self.attn(
            torch.cat((hidden, encoder_outputs), dim=2)
        ))
        attention = self.v(energy).squeeze(2)

        # Convert to probabilities
        return F.softmax(attention, dim=1)

class AttentionDecoder(nn.Module):
    def __init__(self, output_size, hidden_size, attention):
        """Decoder with attention mechanism"""
        super().__init__()
        self.attention = attention
        self.lstm = nn.LSTM(hidden_size * 2, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
```

How Attention Works:

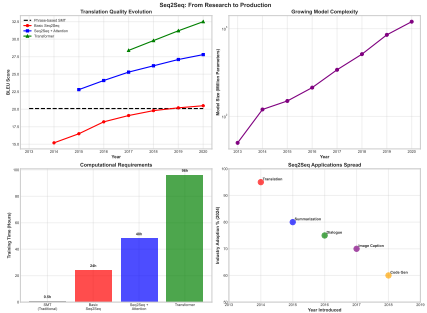
- Score each encoder state
- Softmax creates distribution
- Weighted sum of encoder states
- Concatenate with decoder state

Performance Boost:

- BLEU: 34.8 $\rightarrow$ 41.8[10]
- Handles 2x longer sequences
- Interpretable alignments

15% improvement with attention

Seq2Seq Impact: Translation Quality Leap



Key Insights

- Phrase-based SMT dominated for decades
- Seq2seq matched SMT in 2014
- Attention surpassed all previous methods
- Google deployed in production 2016
- Now foundation for all translation systems

Beyond Translation: Seq2Seq Everywhere (2024)

Text Generation:

- Email auto-complete[11]
- Code documentation
- Story continuation
- Dialogue systems

Summarization:

- News → headlines
- Papers → abstracts
- Meetings → notes
- Videos → descriptions

Data Transformation:

- Text → SQL queries[12]
- Natural language → code
- Speech → text
- Image → caption

Still Competitive:

- Smaller than transformers
- Streaming applications
- Low-latency requirements
- Resource-constrained devices

Seq2seq principle: Any input → any output transformation

[1] Gmail Smart Compose uses seq2seq variants

[2] Text-to-SQL still uses specialized seq2seq models

Week 4 Exercise: Build a Smart Chatbot

Your Mission: Create a chatbot that gives variable-length responses

Example Behavior:

- "Hi" → "Hello! How can I help you today?"
- "What's the weather?" → "I'd need to check current conditions. What's your location?"
- "Tell me a joke" → (Generates different length jokes)

Implementation Steps:

1. Build encoder-decoder architecture
2. Train on conversation pairs
3. Implement attention mechanism
4. Compare responses with/without attention
5. Visualize what words the model attends to

Bonus Challenges:

- Implement beam search (top-k responses)
- Add personality tokens (formal/casual)
- Handle multi-turn conversations
- Measure response diversity

You'll discover: Why chatbots sometimes give generic responses!

Key Takeaways: Breaking Free from Fixed Length

What we learned:

- Language tasks need variable-length input/output
- Encoder-decoder separates understanding from generation
- Context vector creates information bottleneck
- Attention lets decoder access all input information
- Seq2seq enables any-to-any sequence transformation

The evolution:

Fixed I/O (RNN) $\rightarrow$ Variable I/O (Seq2seq) $\rightarrow$ Full visibility (Attention)

Critical Innovation: Attention shows us what the model is "thinking" - interpretability!

Next week: The Transformer
What if we used ONLY attention, no RNNs at all?

References and Further Reading

Foundational Papers:

- Sutskever et al. (2014). "Sequence to Sequence Learning with Neural Networks", NeurIPS
- Bahdanau et al. (2015). "Neural Machine Translation by Jointly Learning to Align and Translate"
- Cho et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder"

Applications:

- Wu et al. (2016). "Google's Neural Machine Translation System"
- See et al. (2017). "Get To The Point: Summarization with Pointer-Generator Networks"
- Vinyals & Le (2015). "A Neural Conversational Model"

Recommended Resources:

- TensorFlow Seq2seq Tutorial (with attention visualization)
- Visualizing and Understanding Neural Machine Translation
- PyTorch Chatbot Tutorial