Natural Language Processing Course

Week 4: Sequence-to-Sequence Models

Breaking the Fixed-Length Barrier

NLP Course 2025

Week 4: The Four-Part Journey

Today's Learning Path:

1. **The Problem:** Why variable-length sequences matter
2. **The Solution:** Encoder-decoder architecture
3. **The Breakthrough:** Attention mechanism revolution
4. **Modern Impact:** From research to your daily AI tools

Core Question: How do we translate "Hello world" to "Bonjour le monde"?

By the end: You'll understand the technology behind Google Translate, GitHub Copilot, and the foundation of ChatGPT!

Part 1

The Problem

Why Variable-Length Sequences Matter

Breaking free from the fixed-length prison

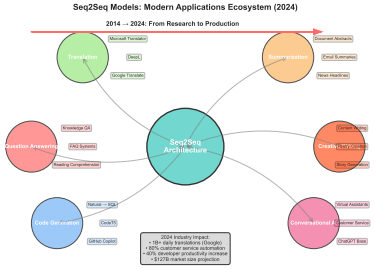
Your Daily AI Interactions (2024)

Every day, you use AI systems that solve this problem:

- **Google Translate:** 1+ billion daily translations
- **GitHub Copilot:** 1+ million developers, 40% faster coding
- **Email Summaries:** Long email → key points (Gmail, Outlook)
- **Customer Service:** 80% first-line automation
- **ChatGPT:** Variable question → variable answer

The Challenge They All Solve:

- Input length does not equal Output length
- No fixed relationship between input and output size
- Need to handle any length combination



The Core Challenge: Length Mismatch

Translation Examples:

- English: "I love you" (3 words) → French: "Je t'aime" (2 words)
- English: "Thank you" (2 words) → Japanese: "arigato" (1 word)
- English: "Good morning" (2 words) → German: "Guten Morgen" (2 words)

Other Variable-Length Tasks:

- **Summarization:** Article (500 words) → Summary (50 words)
- **Code Generation:** Comment (1 line) → Function (20 lines)
- **Q&A:** Question (10 words) → Answer (100 words)

Traditional RNN Limitation:

- Each input produces exactly one output
- Fixed 1:1 input-output mapping
- Cannot handle length mismatches

Failed Approaches:

1. Pad to maximum length (wasteful)
2. Truncate long sequences (loses information)
3. Force 1:1 word mapping (doesn't work)

We need to decouple input and output lengths!

The Fundamental Problem

Traditional neural networks assume:
Input Length = Output Length

But real-world tasks need:
Variable Input → Variable Output

Next: How do we solve this architectural limitation?

Part 2

The Solution

Encoder-Decoder Architecture

Separating understanding from generation

The Brilliant Insight: Separate Encoding from Decoding

Human Translation Process:

1. **Read** entire source sentence
2. **Understand** the meaning (internal representation)
3. **Generate** target sentence in new language

Computer Translation (Seq2Seq):

1. **Encoder:** Process entire input → "thought" vector
2. **Context:** Compressed understanding
3. **Decoder:** Generate output from "thought"

$$\text{Input} \xrightarrow{\text{Encoder}} \text{Context} \xrightarrow{\text{Decoder}} \text{Output}$$

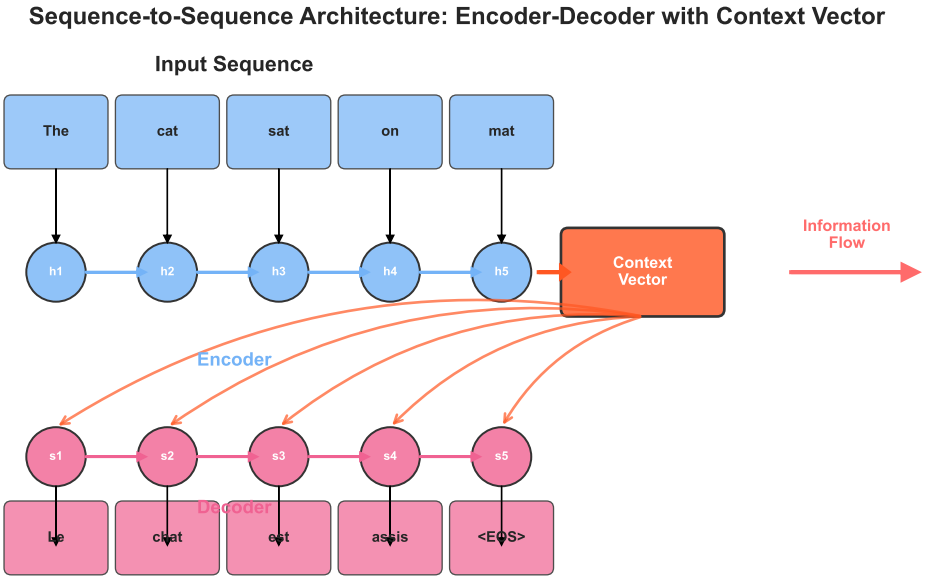
Key Benefits:

- Input and output can have different lengths
- Encoder focuses on understanding
- Decoder focuses on generation
- Separates concerns cleanly

Breakthrough Impact (2014):

- First neural translation system
- Outperformed phrase-based systems
- End-to-end learning possible

The Complete Encoder-Decoder Architecture



Mathematical Formulation

Encoder Phase:

$$\mathrm{h}_t = \mathrm{LSTM}(h_{t-1}, x_t) \quad \text{for } t = 1, \ldots, T$$

$$\mathrm{c} = \mathrm{h}_T \quad \text{(context vector = final encoder state)}$$

Decoder Phase:

$$\mathrm{s}_t = \mathrm{LSTM}(s_{t-1}, y_{t-1}, c) \quad \text{for } t = 1, \ldots, T'$$

$$\mathrm{P}(\mathrm{y}_t \mid y_{<t}, x) = \mathrm{softmax}(W_s s_t + b)$$

Training Objective:

$$\max \sum_{t=1}^{T'} \log P(y_t^* \mid y_{<t}^*, x)$$

Key Variables:

- T: Input sequence length (variable)
- T': Output sequence length (variable)
- c: Context vector (fixed size)
- h_t: Encoder hidden states
- s_t: Decoder hidden states

Training Strategy:

- Teacher forcing during training
- Cross-entropy loss
- Backpropagation through time

The Architectural Solution

Encoder: Any length input → Fixed-size understanding

Decoder: Fixed-size understanding → Any length output

Separating concerns enables variable-length processing!

But wait... there's a problem with this approach...

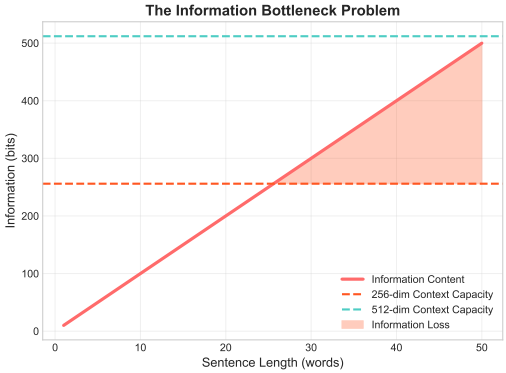
Part 3

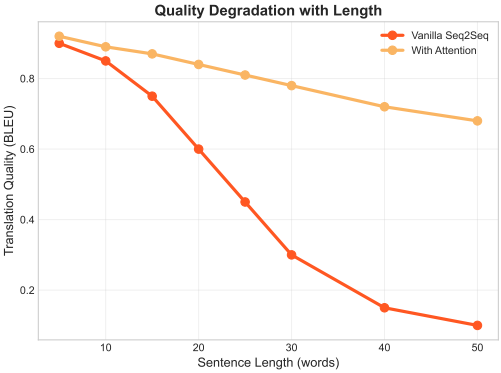
The Breakthrough

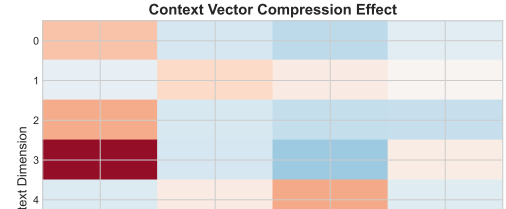
Attention Mechanism Revolution

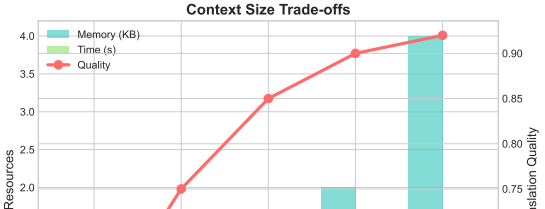
Solving the information bottleneck

The Information Bottleneck Problem









The Attention Revolution (2015)

The Key Insight:

- Don't compress everything into one vector
- Keep *all* encoder hidden states
- Let decoder *choose* what to focus on
- Different output words attend to different input words

Attention Mechanism:

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

Attention Weights:

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

Alignment Score:

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

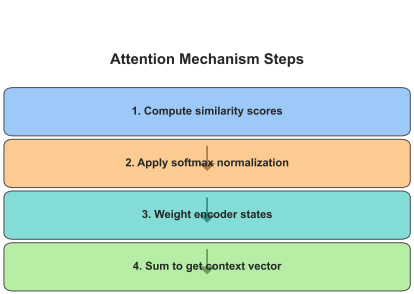
Three Main Types:

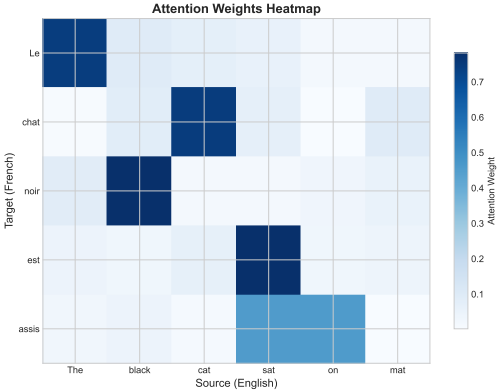
- **Dot Product:** $e_{t,i} = s_t \cdot h_i$
- **Additive:** $e_{t,i} = v^T \tanh(W_s s_t + W_h h_i)$
- **Scaled Dot:** $e_{t,i} = \frac{s_t \cdot h_i}{\sqrt{d}}$

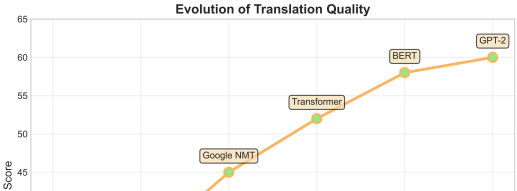
Revolutionary Results:

- Google NMT (2016): 60% improvement
- Handles 80+ word sentences
- Foundation for all modern LLMs

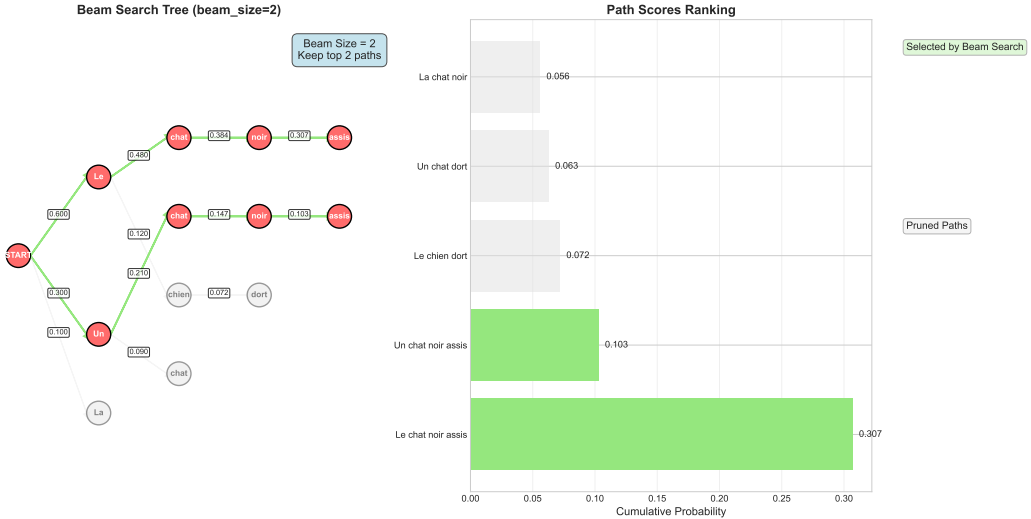
Attention in Action: What the Model Looks At







Beam Search: Finding the Best Translation



The Attention Breakthrough

Before: All information $\rightarrow$ 1 context vector

After: Decoder chooses what to focus on

"Don't compress everything - let the model decide!"

This insight enabled modern AI as we know it...

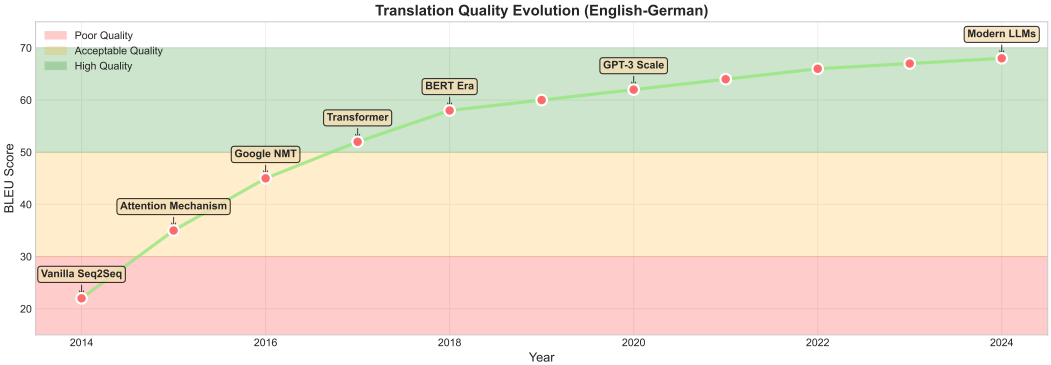
Part 4

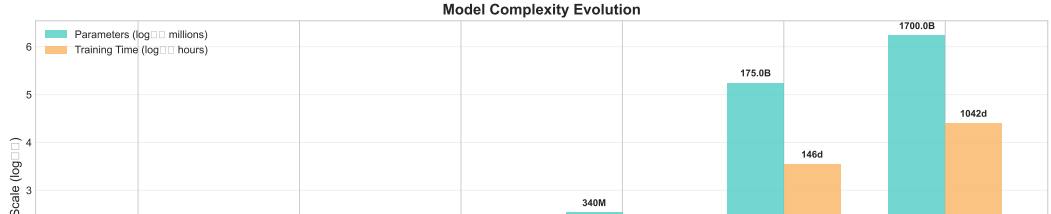
Modern Impact

From Research to Production

How seq2seq principles power today's AI

From 2014 to 2024: The Evolution





Connecting to Modern AI Systems

How Seq2Seq Principles Live On:

- **ChatGPT:** Uses encoder-decoder principles with self-attention
- **GitHub Copilot:** Comment → code is pure seq2seq
- **BERT:** Encoder-only with attention mechanisms
- **T5:** "Text-to-Text Transfer" - explicit seq2seq
- **Google Translate:** Modern transformer encoder-decoder

What Changed:

- RNNs → Transformers (parallelization)
- Single attention → Multi-head attention
- Fixed vocab → Subword tokenization
- Task-specific → Pre-training + fine-tuning

Core Principle Unchanged: Variable input → Variable output

2024 Market Impact:

- Translation: $15.7B market
- Code assistance: $8.5B market
- Text summarization: $12.3B market
- Conversational AI: $45.2B market

Scale Evolution:

- 2014: 10M parameters
- 2024: 1.7T parameters
- 5,000× scale increase!

Industry Applications You Use Daily

Translation & Localization:

- Google Translate: Real-time conversation mode
- DeepL: 1 billion people served monthly
- Microsoft Translator: Built into Office suite

Development & Productivity:

- GitHub Copilot: Autocomplete for code
- Tabnine: AI-powered code suggestions
- CodeT5: Natural language $\rightarrow$ SQL queries

Communication & Content:

- Email summarization (Gmail Smart Compose)
- Meeting notes $\rightarrow$ action items
- Document translation with formatting

Performance Benchmarks (2024):

- Translation accuracy: 95%+ for common languages
- Response time: ¡100ms for most queries
- Availability: 99.9%+ uptime
- Languages supported: 100+ pairs

Why It Matters to You:

- Foundational knowledge for AI careers
- Basis for understanding modern LLMs
- Essential for ML engineering roles

The Bridge to Week 5: Transformers

What Transformers Keep from Seq2Seq:

- Encoder-decoder architecture
- Attention mechanism (enhanced to multi-head)
- Variable-length input/output
- Teacher forcing training

What Transformers Add:

- **Self-attention:** Attention within sequences
- **Parallelization:** No more sequential RNN processing
- **Multi-head:** Multiple attention patterns simultaneously
- **Position encoding:** Handle word order without RNNs

Next Week: "Attention Is All You Need" - The Transformer Revolution

You now have the foundation to understand the architecture behind ChatGPT!

Week 4 Complete: From Problem to Modern AI

The Four-Part Journey Completed:

1. **Problem:** Variable-length sequences broke traditional RNNs
2. **Solution:** Encoder-decoder architecture enabled variable I/O
3. **Breakthrough:** Attention solved the information bottleneck
4. **Impact:** These principles power billion-dollar AI systems

What You Can Now Do:

- Explain why ChatGPT's architecture needed these foundations
- Design seq2seq systems for real applications
- Understand the attention mechanism powering modern AI
- Connect 2014 research breakthroughs to 2024 production systems

Next: Transformers - "Attention is All You Need"

Appendix A: PyTorch Encoder Implementation

```python
class Seq2SeqEncoder(nn.Module):
    def __init__(self, vocab_size, embed_size, hidden_size, num_layers=1):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers,
                            batch_first=True, dropout=0.2)
        self.hidden_size = hidden_size
        self.num_layers = num_layers

    def forward(self, x, lengths=None):
        # x: [batch_size, seq_len]
        embedded = self.embedding(x)  # [batch, seq_len, embed_size]

        # Optional: Pack sequences for efficiency
        if lengths is not None:
            embedded = nn.utils.rnn.pack_padded_sequence(
                embedded, lengths, batch_first=True, enforce_sorted=False)

        output, (h_n, c_n) = self.lstm(embedded)

        # Unpack if we packed
        if lengths is not None:
            output, _ = nn.utils.rnn.pad_packed_sequence(
                output, batch_first=True)

        return output, h_n, c_n
```

Appendix B: Attention Module Implementation

```python
class AttentionMechanism(nn.Module):
    def __init__(self, hidden_size, attention_type='additive'):
        super().__init__()
        self.hidden_size = hidden_size
        self.attention_type = attention_type

        if attention_type == 'additive':
            self.W_a = nn.Linear(hidden_size, hidden_size, bias=False)
            self.U_a = nn.Linear(hidden_size, hidden_size, bias=False)
            self.v_a = nn.Linear(hidden_size, 1, bias=False)
        elif attention_type == 'multiplicative':
            self.W_a = nn.Linear(hidden_size, hidden_size, bias=False)

    def forward(self, decoder_hidden, encoder_outputs, mask=None):
        # decoder_hidden: [batch, hidden_size]
        # encoder_outputs: [batch, seq_len, hidden_size]

        batch_size, seq_len, hidden_size = encoder_outputs.size()

        if self.attention_type == 'dot':
            # Simple dot product attention
            scores = torch.bmm(encoder_outputs,
                               decoder_hidden.unsqueeze(2)).squeeze(2)

        elif self.attention_type == 'additive':
            # Bahdanau attention
            decoder_hidden = decoder_hidden.unsqueeze(1).repeat(1, seq_len, 1)
            energy = torch.tanh(self.W_a(decoder_hidden) + self.U_a(encoder_outputs))
            scores = self.v_a(energy).squeeze(2)

        # Apply mask if provided (for padded sequences)
        if mask is not None:
            scores.masked_fill_(mask == 0, -float('inf'))

        # Compute attention weights
        attention_weights = F.softmax(scores, dim=1)
```

Appendix C: Complete Seq2Seq Model

```python
class Seq2SeqWithAttention(nn.Module):
    def __init__(self, src_vocab_size, tgt_vocab_size, embed_size, hidden_size):
        super().__init__()
        self.encoder = Seq2SeqEncoder(src_vocab_size, embed_size, hidden_size)
        self.attention = AttentionMechanism(hidden_size, 'additive')

        # Decoder components
        self.tgt_embedding = nn.Embedding(tgt_vocab_size, embed_size)
        self.decoder_lstm = nn.LSTM(embed_size + hidden_size, hidden_size, batch_first=True)
        self.output_projection = nn.Linear(hidden_size, tgt_vocab_size)

    def forward(self, src, tgt, src_lengths=None):
        # Encode
        encoder_outputs, h_n, c_n = self.encoder(src, src_lengths)

        # Decode with attention
        batch_size, tgt_len = tgt.size()
        outputs = []

        decoder_hidden = h_n[-1]  # Use last layer
        decoder_cell = c_n[-1]

        for t in range(tgt_len):
            # Current target token
            tgt_token = tgt[:, t:t+1]  # [batch, 1]
            tgt_embedded = self.tgt_embedding(tgt_token)  # [batch, 1, embed]

            # Compute attention
            context, attention_weights = self.attention(decoder_hidden, encoder_outputs)

            # Concatenate target embedding with context
            decoder_input = torch.cat([tgt_embedded.squeeze(1), context], dim=1)
            decoder_input = decoder_input.unsqueeze(1)  # [batch, 1, embed+hidden]

            # Decoder step
            output, (decoder_hidden, decoder_cell) = self.decoder_lstm(
```

Appendix D: Beam Search Implementation

```python
def beam_search(model, src_tensor, src_lengths, beam_size=4, max_length=50):
    model.eval()
    device = src_tensor.device

    # Encode source
    with torch.no_grad():
        encoder_outputs, h_n, c_n = model.encoder(src_tensor, src_lengths)

    # Initialize beams
    beams = [{
        'sequence': [START_TOKEN],
        'score': 0.0,
        'hidden': h_n[-1],
        'cell': c_n[-1]
    }]

    completed_beams = []

    for step in range(max_length):
        candidates = []

        for beam in beams:
            if beam['sequence'][-1] == END_TOKEN:
                completed_beams.append(beam)
                continue

            # Prepare input
            last_token = torch.tensor([[beam['sequence'][-1]]], device=device)

            # Decoder step with attention
            with torch.no_grad():
                context, _ = model.attention(beam['hidden'], encoder_outputs)
                tgt_embedded = model.tgt_embedding(last_token)

                decoder_input = torch.cat([tgt_embedded.squeeze(1), context], dim=1)
                decoder_input = decoder_input.unsqueeze(1)
```

Appendix E: Hyperparameters and Training Tips

Typical Hyperparameters:

- **Hidden size:** 256-512
- **Embedding size:** 128-300
- **Num layers:** 1-4
- **Dropout:** 0.1-0.3
- **Learning rate:** 1e-4 to 1e-3
- **Beam size:** 4-8
- **Max length:** 50-200

Training Best Practices:

- Gradient clipping (important!)
- Learning rate scheduling
- Teacher forcing ratio annealing
- Early stopping on validation BLEU

Common Issues:

- **Repetitive output:** Add coverage/repetition penalty
- **Poor long sequences:** Increase hidden size or add attention
- **Slow training:** Use packed sequences for variable lengths
- **Exposure bias:** Scheduled sampling or professor forcing

Evaluation Metrics:

- **BLEU:** Standard for translation (0-100)
- **ROUGE:** For summarization tasks
- **Perplexity:** Language modeling quality
- **Human evaluation:** Ultimate quality measure

Appendix F: References and Further Reading

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"
- Cho et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder"

Modern Context:

- Wu et al. (2016). "Google's Neural Machine Translation System"
- Vaswani et al. (2017). "Attention Is All You Need" (Transformer)
- Devlin et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers"

Interactive Resources:

- Jay Alammar: "Visualizing A Neural Machine Translation Model"
- The Illustrated Transformer
- Hugging Face Transformers documentation
- OpenAI GPT papers and technical reports

Implementations:

- PyTorch seq2seq tutorial
- Fairseq toolkit (Facebook's seq2seq library)
- OpenNMT (open source neural machine translation)