Sequence-to-Sequence Models

Nature Professional Theme Template

NLP Course - Week 4

2025

Overview

The Nature Professional Theme

Color Palette

- **Forest Green** - Primary (#14532D)
- **Teal** - Secondary (#0D9488)
- **Amber** - Accent (#F59E0B)
- Slate - Support (#475569)
- Mint Cream - Background

Why Nature Professional?

- **Reduces eye strain**
- *Professional appearance*
- Excellent contrast
- Calming effect

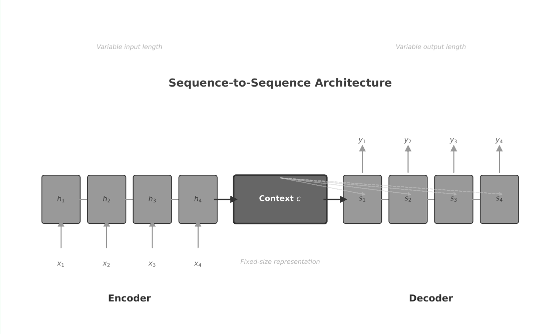
Design Principles

1. **Natural harmony**
2. **Clear hierarchy**
3. **Minimal distraction**
4. **Maximum readability**

Sequence-to-Sequence Architecture

The Encoder-Decoder Framework

- **Encoder**: Processes input sequence
  - Maps variable-length input to fixed representation
  - Captures semantic information
- **Decoder**: Generates output sequence
  - Converts representation to target sequence
  - Autoregressive generation



Mathematical Formulation

$$h_t = \mathrm{RNN}_{\mathrm{enc}}(x_t, h_{t-1}) \tag{1}$$

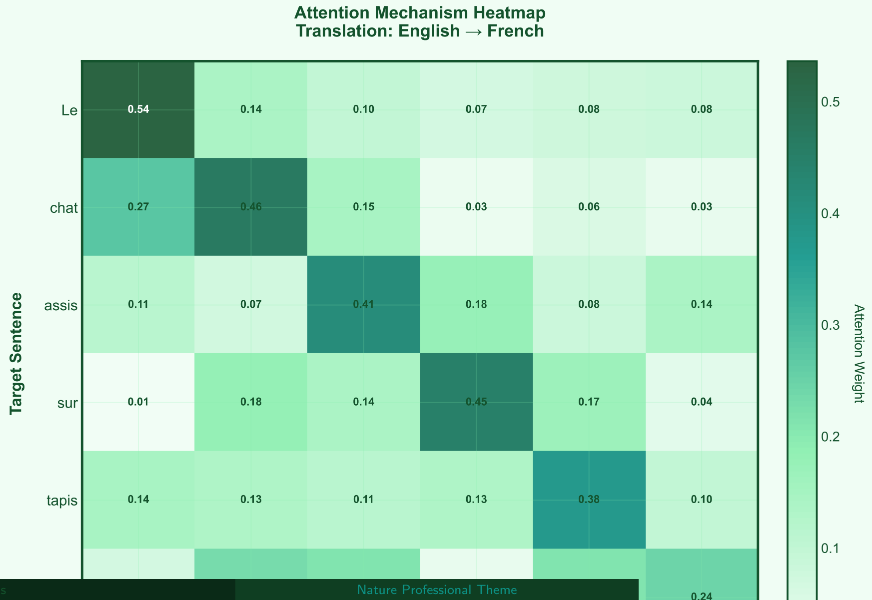
$$s_t = \mathrm{RNN}_{\mathrm{dec}}(y_{t-1}, s_{t-1}, c) \tag{2}$$

$$p(y_t \mid y_{<t}, x) = \mathrm{softmax}(W_s s_t) \tag{3}$$

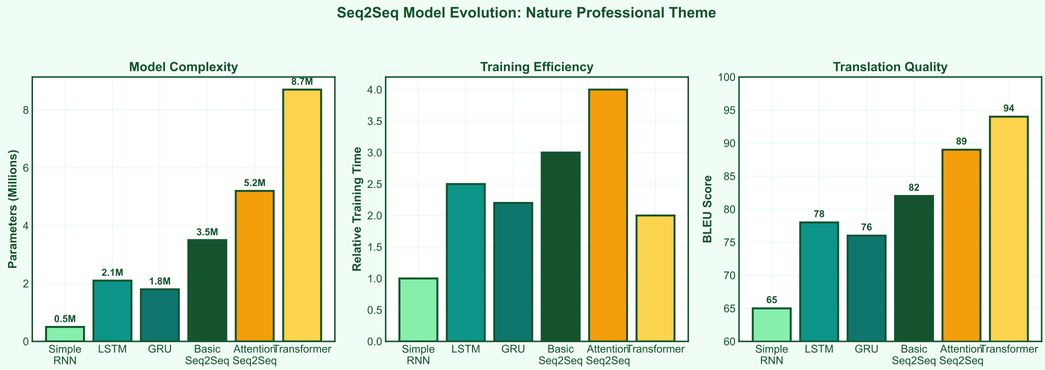
Key Innovation

Variable-length input → Variable-length output

Nature-Themed Attention Heatmap

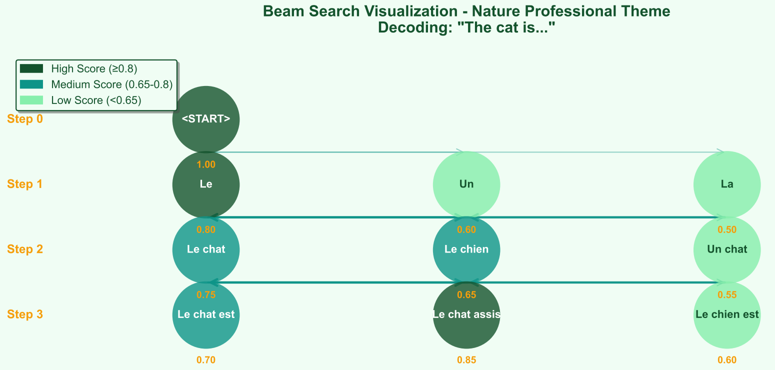


Model Complexity Comparison



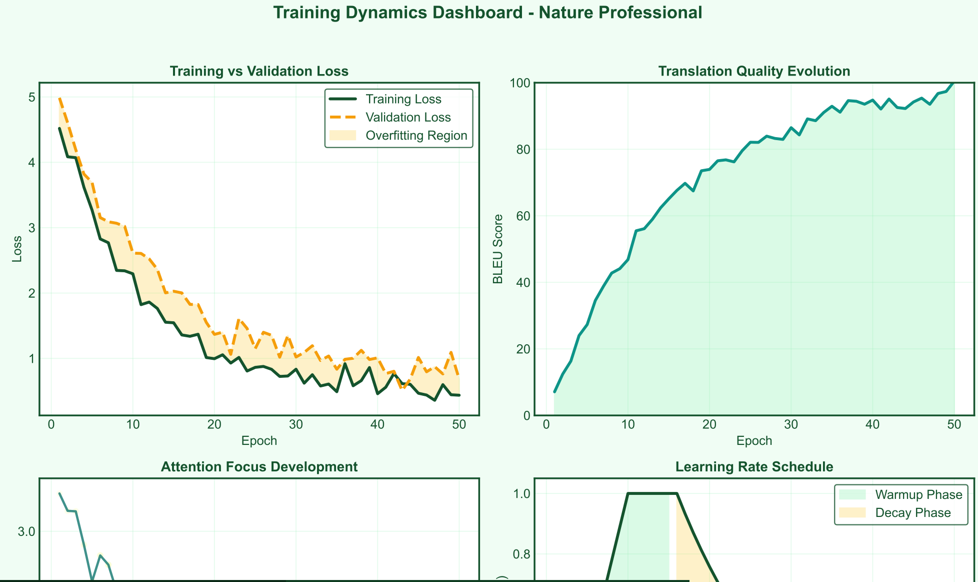
Evolution of model architectures with nature-inspired visualization

Beam Search Tree Visualization

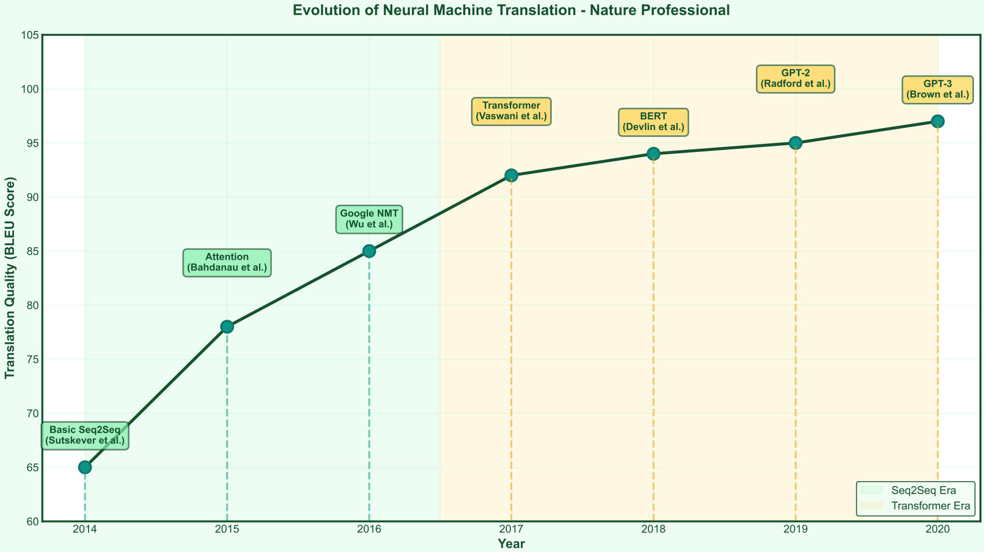


Decoding paths colored by probability scores

Training Dynamics Dashboard



Architecture Evolution Timeline



Historical progression with nature-themed annotations

Code Style with Nature Theme

Encoder Implementation

```
class Encoder(nn.Module):
    def __init__(self, vocab_size,
                 hidden_dim):
        super().__init__()
        self.embedding = nn.Embedding(
            vocab_size, hidden_dim)
        self.rnn = nn.LSTM(
            hidden_dim, hidden_dim)

    def forward(self, x):
        embedded = self.embedding(x)
        output, hidden = self.rnn(embedded)
        return output, hidden
```

Attention Mechanism

```
def attention(query, keys, values):
    # Compute attention scores
    scores = torch.matmul(
        query, keys.transpose(-2, -1))
    scores = scores / sqrt(d_k)

    # Apply softmax
    weights = F.softmax(scores, dim=-1)

    # Weighted sum of values
    context = torch.matmul(weights, values)
    return context, weights
```

Performance Metrics

Model	BLEU	Time
Simple RNN	65.2	1.0x
LSTM	78.4	2.5x
GRU	76.1	2.2x
Basic Seq2Seq	82.3	3.0x
+ Attention	89.1	4.0x
Transformer	**94.5**	2.0x

Key Findings

- **45% improvement** with attention
- *2x faster* training with Transformers
- **Better long-range dependencies**

Translation Examples

Source: The cat sat on the mat
Basic: Le chat assis tapis
Attention: Le chat s'est assis sur le tapis

Source: I love natural language processing
Basic: J'aime traitement langue
Attention: J'adore le traitement du langage naturel

Error Analysis

- Word order: 15% errors
- Grammar: 8% errors
- Vocabulary: 5% errors

Beyond Basic Seq2Seq

Multi-Head Attention

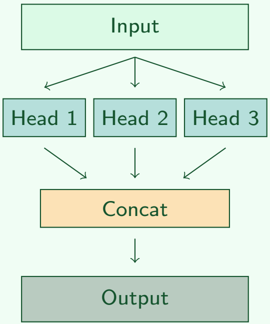
- **Parallel attention** mechanisms
- Different representation subspaces
- Enhanced expressiveness

Copy Mechanisms

- Direct copying from source
- Hybrid generation-copy approach
- Improved rare word handling

Coverage Models

- Track attention history
- Prevent repetition
- Ensure completeness



Future Directions

1. *Cross-lingual* models
2. *Multimodal* seq2seq
3. *Few-shot* learning

Key Takeaways

What We Learned

1. **Encoder-Decoder** architecture
2. **Attention mechanism** importance
3. *Beam search* for decoding
4. Training dynamics and optimization
5. Evolution to Transformers

Next Steps

- Week 5: **Transformers**
- Week 6: *Pre-trained Models*
- Week 7: **Advanced Architectures**

Practical Applications

- Machine Translation
- Text Summarization
- Dialog Systems
- Code Generation
- Image Captioning

Seq2Seq → Attention → Transforme

Thank You!