Sequence-to-Sequence Models

Week 4: The Translation Revolution with Attention

NLP Course 2025

Professional Template Edition

September 29, 2025

Week 4: Journey Through Translation and Attention

Learning Path: From word-by-word replacement to neural translation. Master encoder-decoder architectures, understand the bottleneck problem, and discover how attention revolutionized machine translation.

Part 1: Translation Challenge & Motivation

Why Word-by-Word Translation Fails

The Google Translate Evolution: A Success Story

2006: Statistical MT

- Word/phrase dictionaries
- Counted co-occurrences
- "Reasonable" translations
- Often awkward phrasing

2016: Neural MT Launch

- Seq2Seq with attention
- Human-quality for some pairs
- 60% error reduction
- Revolutionary improvement

Real Example:

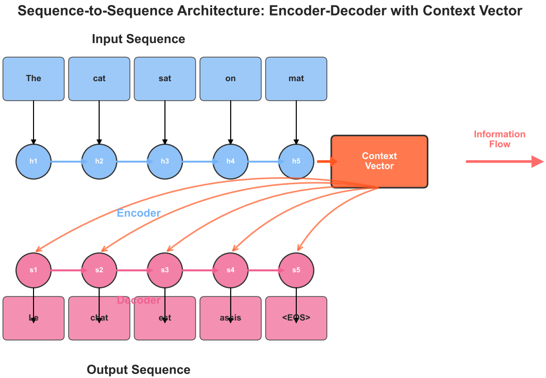
Chinese Input: "There is one cat in station"

Old: "In the station is one cat"
New: "There is a cat at the station"

What changed? Understanding context, not just words

Historical Context: Neural MT reduced translation errors by 60% overnight - the biggest leap in MT history

The Fundamental Problem: Meaning Across Languages



Translation is NOT:

- Word replacement
- Dictionary lookup
- Rule application
- Pattern matching

Translation IS:

- Understanding meaning
- Cultural context
- Reformulation
- Creative generation

Why Word-by-Word Translation Fails: Concrete Examples

Problem 1: Word Order

- English: "I saw the red house"
- Spanish: "Vi la casa roja"
- Literal: "Saw-I the house red"

Problem 2: Idioms

- English: "It's raining cats and dogs"
- French: "Il pleut des cordes"
- Literal: "It rains ropes"

Problem 3: Context

- "Bank" $\rightarrow$ "Banque" (financial)
- "Bank" $\rightarrow$ "Rive" (river)
- Need full sentence to decide

Problem 4: Grammar

- German: Verb at end
- Japanese: Subject optional
- Chinese: No tenses

Conclusion: Languages encode meaning differently - translation needs deep understanding

Language Diversity: Each language has unique ways of expressing ideas

Converting Meaning to Numbers: The Core Challenge

Computers only understand numbers, so:

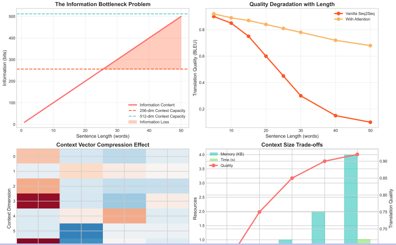
"The cat sat on the mat" $\rightarrow$ **[Numbers]** $\rightarrow$ "Le chat s'est assis sur le tapis"

Step 1: Words to Vectors

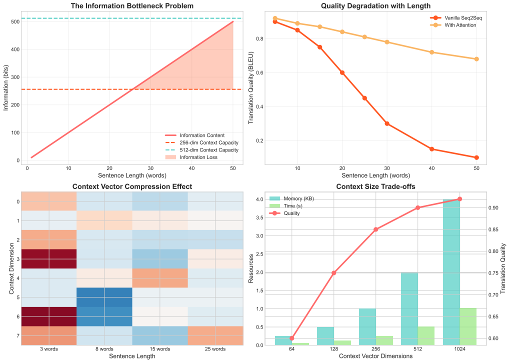
- "cat" $\rightarrow$ [0.2, -0.5, 0.8, ...]
- 100-300 dimensional vectors
- Learned from context (Word2Vec)
- Similar words = nearby vectors

Step 2: Sentence to Vector

- Combine word vectors
- Build "context vector"
- Fixed size (e.g., 256 dims)
- Must capture ALL meaning



The Compression Challenge: Information Bottleneck



Compression Ratios:

- 5 words: 500 dims → 256 (2:1)
- 20 words: 2000 dims → 256 (8:1)
- 50 words: 5000 dims → 256 (20:1)

Problem: More compression = More loss

What Gets Lost?

- Specific word choices
- Grammatical nuances
- Word positions
- Long-range dependencies

Interactive Exercise: Manual Translation Steps

Task: Translate "The black cat sat" to French step-by-step

Your Steps:

1. Read entire English sentence
2. Identify: subject (cat), verb (sat)
3. Recall French words:
   - cat → chat
   - black → noir
   - sat → s'est assis
4. Apply French grammar:
   - Article-Noun-Adjective order
   - Gender agreement (le/la)
5. Generate: "Le chat noir s'est assis"

What You Actually Did:

1. Encoded English to meaning
2. Stored meaning in memory
3. Decoded meaning to French

This is exactly Seq2Seq!

Key Observation:
You didn't translate word-by-word! You understood first, then generated.

Human Insight: We naturally use encoder-decoder approach when translating

Calculating the Bottleneck: A Mathematical Perspective

Information Content:

$$\text{Input} = n \times d_{\text{embed}} \quad (1)$$

$$\text{Context} = d_{\text{hidden}} \quad (2)$$

$$\text{Ratio} = \frac{n \times d_{\text{embed}}}{d_{\text{hidden}}} \quad (3)$$

Example Calculation:

- 20 words, 100-dim embeddings
- Input: $20 \times 100 = 2000$ values
- Context: 256 values
- Compression: $\frac{2000}{256} \approx 8:1$

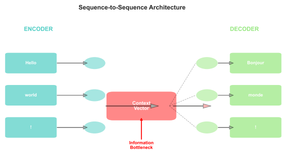
The Problem:
Cannot fit 2000 numbers into 256 without loss!

Mathematical Reality: Information theory limits how much we can compress without loss

Part 1 Summary: Understanding the Challenge

What We Learned:

- Translation $\neq$ word replacement
- Languages encode differently
- Need meaning understanding
- Must convert to numbers
- Fixed-size bottleneck problem



The Challenge:

- Variable input length
- Fixed context size
- Information loss inevitable
- Longer = worse compression

Key Question:
How do we capture all meaning in a fixed-size vector?

Next: The encoder-decoder architecture - a first solution to the translation challenge

Part 2: Encoder-Decoder Architecture

Building Understanding, Then Generating

The Two-Phase Translation Intuition

How humans translate (simplified):

Phase 1: Understanding

1. Read entire source sentence
2. Extract complete meaning
3. Store in "mental representation"
4. Forget specific words
5. Keep abstract meaning

Result: Language-agnostic meaning

Phase 2: Generation

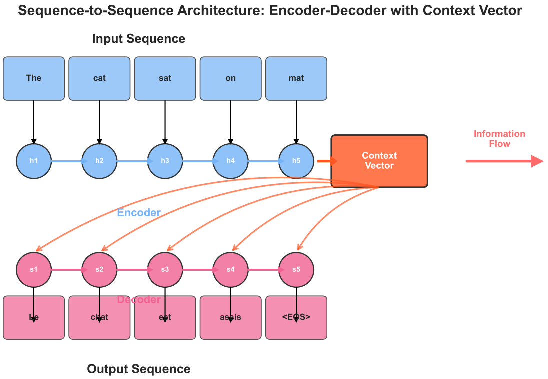
1. Access stored meaning
2. Apply target grammar
3. Choose appropriate words
4. Generate word-by-word
5. Maintain coherence

Result: Natural target sentence

Neural Equivalent: Encoder (understanding) + Decoder (generation) = Seq2Seq

Cognitive Model: Seq2Seq mimics human two-phase translation process

The Encoder: Building Understanding Step-by-Step



Encoder's Job:

- Process input sequentially
- Build hidden state (memory)
- Update with each word
- Final state = full understanding

Processing "The cat sat":

1. $h_1 = \text{RNN}(\text{"The"}, h_0)$
2. $h_2 = \text{RNN}(\text{"cat"}, h_1)$
3. $h_3 = \text{RNN}(\text{"sat"}, h_2)$
4. Context: $c = h_3$

The Decoder: Generating from Understanding

Decoder's Job:

- Start with context vector c
- Generate one word at a time
- Use previous word + context
- Stop at end token

Generation Process:

$$s_0 = c \text{ (initialize)} \tag{6}$$

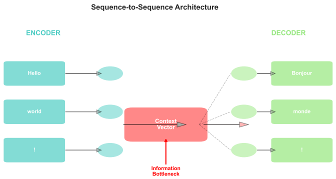
$$s_t = \text{RNN}(y_{t-1}, s_{t-1}) \tag{7}$$

$$P(y_t) = \mathrm{softmax}(Ws_t) \tag{8}$$

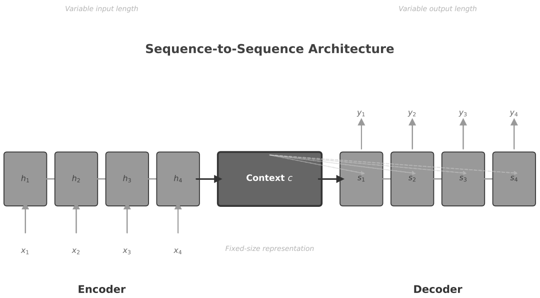
Generating "Le chat noir":

1. Start: $s_0 = c$, $y_0 =$ ¡START¿
2. Generate "Le": $P(y_1 \mid c)$
3. Generate "chat": $P(y_2 \mid y_1, c)$
4. Generate "noir": $P(y_3 \mid y_{1:2}, c)$
5. Stop: $y_4 =$ ¡END¿

Key: Each word depends on context + history



Complete Seq2Seq Architecture



Components:

- Embedding layers
- Encoder RNN
- Context vector
- Decoder RNN

Training:

- Teacher forcing
- Cross-entropy loss
- Backprop through time
- End-to-end learning

Inference:

- Greedy decoding
- Beam search
- Length normalization
- Coverage penalty

Complete Seq2Seq Implementation in PyTorch

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

class Seq2Seq(nn.Module):
    def __init__(self, src_vocab,
                 tgt_vocab, embed_dim=256,
                 hidden_dim=512):
        super().__init__()

        # Embeddings
        self.src_embed = nn.Embedding(
            src_vocab, embed_dim
        )
        self.tgt_embed = nn.Embedding(
            tgt_vocab, embed_dim
        )

        # Encoder & Decoder
        self.encoder = nn.LSTM(
            embed_dim, hidden_dim,
            batch_first=True
        )
        self.decoder = nn.LSTM(
            embed_dim, hidden_dim,
            batch_first=True
        )

        # Output projection
        self.output = nn.Linear(
            hidden_dim, tgt_vocab
```

```python
    def forward(self, src, tgt):
        # Encode
        src_emb = self.src_embed(src)
        _, (h, c) = self.encoder(
            src_emb
        )

        # Decode
        tgt_emb = self.tgt_embed(tgt)
        out, _ = self.decoder(
            tgt_emb, (h, c)
        )

        # Project
        logits = self.output(out)
        return logits

# Usage
model = Seq2Seq(
    src_vocab=10000,
    tgt_vocab=10000
)

# Training step
src = torch.randint(0, 10000,
                    (32, 20))
tgt = torch.randint(0, 10000,
                    (32, 15))
logits = model(src, tgt)
```

Encoding Example: "The black cat sat"

Watch the hidden state evolve:

Step 1: Process "The"

- Input: $x_1 = \text{embed}(\text{"The"}) = [0.1, 0.3, \ldots]$
- Hidden: $h_1 = \text{LSTM}(x_1, h_0)$
- Memory: "Determiner seen"

Step 2: Process "black"

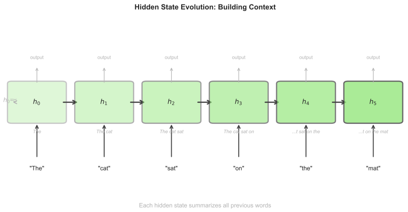
- Input: $x_2 = \text{embed}(\text{"black"})$
- Hidden: $h_2 = \text{LSTM}(x_2, h_1)$
- Memory: "Determiner + adjective"

Step 3: Process "cat"

- Input: $x_3 = \text{embed}(\text{"cat"})$
- Hidden: $h_3 = \text{LSTM}(x_3, h_2)$
- Memory: "Black cat (subject)"

Step 4: Process "sat"

- Input: $x_4 = \text{embed}(\text{"sat"})$
- Hidden: $h_4 = \text{LSTM}(x_4, h_3)$
- Memory: "Black cat sat (complete)"



Final Context:
$c = h_4$ contains: - Subject: black cat - Action: sat - Tense: past

Encoding Process: Each word updates understanding, final state has complete meaning

Decoding Example: Generating “Le chat noir”

Starting from context c:

Step 1: Generate “Le”

- State: $s_0 = c$
- Input: ¡START¿ token
- Output: $P(\text{“Le”}) = 0.8$
- Next: $s_1 = \text{LSTM}(\text{“Le”}, s_0)$

Step 2: Generate “chat”

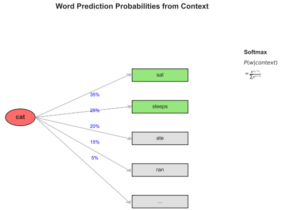
- State: s_1 (knows “Le”)
- Input: “Le”
- Output: $P(\text{“chat”}) = 0.7$
- Next: $s_2 = \text{LSTM}(\text{“chat”}, s_1)$

Step 3: Generate “noir”

- State: s_2 (knows “Le chat”)
- Input: “chat”
- Output: $P(\text{“noir”}) = 0.6$
- Next: $s_3 = \text{LSTM}(\text{“noir”}, s_2)$

Probability Distribution:

At each step, model outputs:



Key Point:
Decoder maintains its own hidden state separate from encoder

Decoding Process: Conditional generation using context and previous outputs

Quiz Checkpoint: Understanding Seq2Seq

Questions:

Q1: What is the context vector?

a) Average of word embeddings
b) Final encoder hidden state
c) Sum of all hidden states
d) Random initialization

Q2: Why use two separate networks?

a) Faster training
b) Different tasks (read vs write)
c) More parameters
d) Requirement of RNNs

Q3: Teacher forcing means:

a) Using true targets during training
b) Forcing convergence
c) Teaching the teacher

Answers:

A1: b) Final encoder hidden state

- Contains full sentence understanding
- Fixed-size representation
- Passed to decoder

A2: b) Different tasks

- Encoder: comprehension
- Decoder: generation
- Different objectives

A3: a) Using true targets

- Feed correct previous word
- Speeds up training
- Avoids error accumulation

Part 2 Summary: The Encoder-Decoder Solution

Architecture Components:

- Encoder RNN: reads input
- Context vector: compressed meaning
- Decoder RNN: generates output
- End-to-end training

Key Equations:

$$c = \text{Encoder}(x_{1:n}) \tag{9}$$

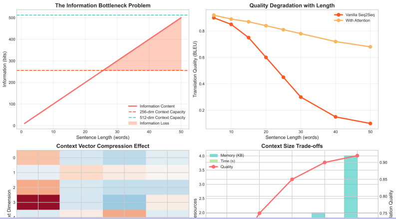
$$y_t = \text{Decoder}(c, y_{<t}) \tag{10}$$

Strengths:

- Variable input/output length
- End-to-end learning
- No alignment needed
- Works for any language pair

Weakness:

Fixed-size bottleneck!



Part 3: The Attention Revolution

Looking Back at All Hidden States

The Bottleneck Problem: Why Seq2Seq Fails on Long Sentences

Performance Degradation:

- 10 words: BLEU = 35
- 20 words: BLEU = 25
- 30 words: BLEU = 15
- 40+ words: BLEU ¡ 10

What's Lost:

- Early words forgotten
- Specific details blurred
- Word positions unclear
- Grammatical structure

Cause: Information bottleneck

Need: Access to all states

Human Translation: The Attention Analogy

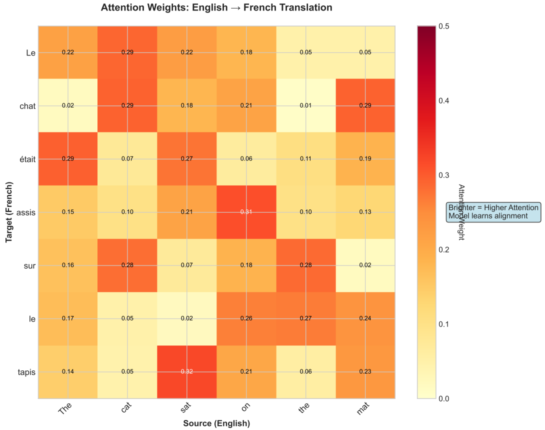
How do humans really translate long sentences?

Translating Word by Word:
"The black cat that I saw yesterday sat"

When translating "sat":

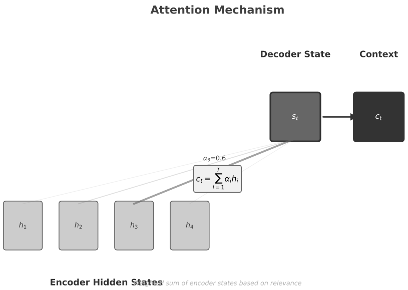
1. Look back at "cat" (subject)
2. Check tense markers
3. Verify agreement
4. Generate appropriate form

Key: We don't memorize everything! We look back as needed.



Attention Idea:

The Attention Mechanism: Dynamic Context



Old Way (Seq2Seq):

- Fixed context $c = h_n$
- Same for all decoder steps
- Information bottleneck
- Forgets early words

New Way (Attention):

- Dynamic context c_t
- Different for each word
- Weighted sum of all states
- Remembers everything

$$c = \sum^{n} \alpha h$$

Attention Mathematics: Computing Weights

How to calculate attention weights:

Step 1: Score How relevant is each encoder state?

$$e_{ti} = \text{score}(s_{t-1}, h_i)$$

Common scoring functions:

- **Dot:** $s_{t-1} \cdot h_i$
- **General:** $s_{t-1} W h_i$
- **Concat:** $v \tanh(W[s_{t-1}; h_i])$

Step 2: Normalize Convert scores to probabilities:

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_j \exp(e_{tj})}$$

Step 3: Weighted Sum Compute dynamic context:

$$c_t = \sum_{i=1}^{n} \alpha_{ti} h_i$$

Each decoder step gets its own custom-weighted view of the source!

Mathematical Core: Three simple steps that revolutionized machine translation

Attention Mathematics: Query-Key-Value Intuition

Understanding the QKV Framework:

Components:

- **Query** (s_{t-1}): What I'm looking for
- **Keys** (h_i): What's available
- **Values** (h_i): What to retrieve
- **Weights** (α_{ti}): Relevance scores

Analogy:

- Query = Search term
- Keys = Document titles
- Values = Document content
- Attention = Search relevance

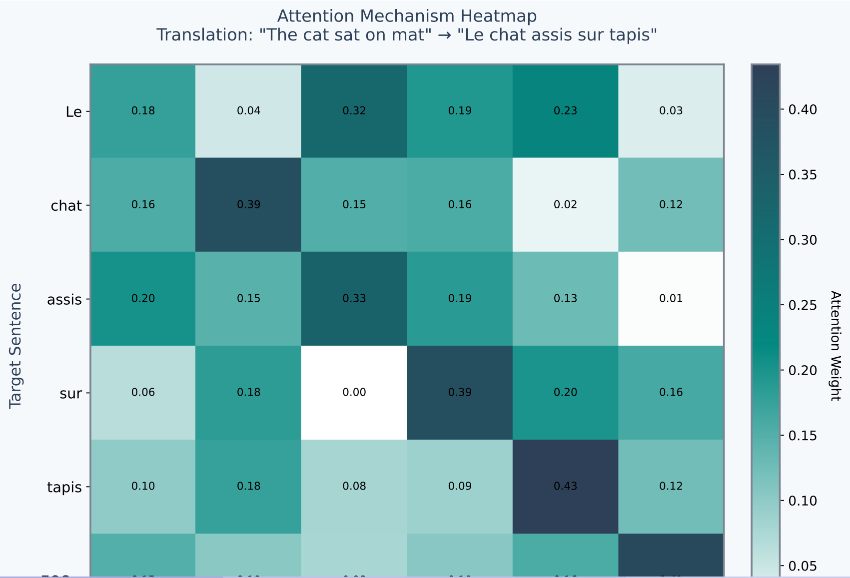
Critical Insight:

This simple mechanism
is the foundation of
ALL modern transformers!

GPT, BERT, T5, ChatGPT...
all use this QKV attention

Foundation: Attention as learned information retrieval - the key to modern AI

Visualizing Attention: What the Model Focuses On



Implementing Attention: Core Logic

```python
class Attention(nn.Module):
    def forward(self, hidden, encoder_outputs):
        # hidden: [1, batch, hidden_dim] - decoder state
        # encoder_outputs: [batch, seq_len, hidden_dim] - all encoder states

        seq_len = encoder_outputs.size(1)

        # Repeat decoder hidden for each encoder position
        hidden = hidden.squeeze(0).unsqueeze(1).repeat(1, seq_len, 1)

        # Score each encoder state (concat scoring)
        energy = torch.tanh(self.attn(
            torch.cat((hidden, encoder_outputs), dim=2)
        ))

        # Convert to scalar scores and apply softmax
        attention = self.v(energy).squeeze(2)
        weights = F.softmax(attention, dim=1)

        # Weighted sum of encoder states
        context = torch.bmm(weights.unsqueeze(1), encoder_outputs)

        return context, weights
```

Core Implementation: Concat scoring function with learned parameters

Implementing Attention: Usage in Decoder

```python
class DecoderWithAttention(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.attention = Attention(hidden_dim)
        self.lstm = nn.LSTM(embed_dim + hidden_dim, hidden_dim)
        self.output = nn.Linear(hidden_dim, vocab_size)

    def forward(self, input_token, hidden, encoder_outputs):
        # Embed current token
        embedded = self.embedding(input_token)  # [batch, 1, embed]

        # Compute attention context
        context, attn_weights = self.attention(hidden, encoder_outputs)
        # context: [batch, 1, hidden_dim]

        # Concatenate embedding with context
        lstm_input = torch.cat((embedded, context), dim=2)

        # LSTM step with attention-augmented input
        output, hidden = self.lstm(lstm_input, hidden)

        # Project to vocabulary
        predictions = self.output(output)

        return predictions, hidden, attn_weights

# Training loop usage
decoder = DecoderWithAttention(vocab_size=10000, embed_dim=256, hidden_dim=512)
```

Interactive Exercise: Calculate Attention Weights

Task: Compute attention for generating “noir” (black)

Given decoder state s_2 after generating “Le chat”:

Encoder states:

- h_1: “The” $= [0.1, 0.2]$
- h_2: “black” $= [0.8, 0.9]$
- h_3: “cat” $= [0.5, 0.4]$
- h_4: “sat” $= [0.3, 0.1]$

Decoder query:

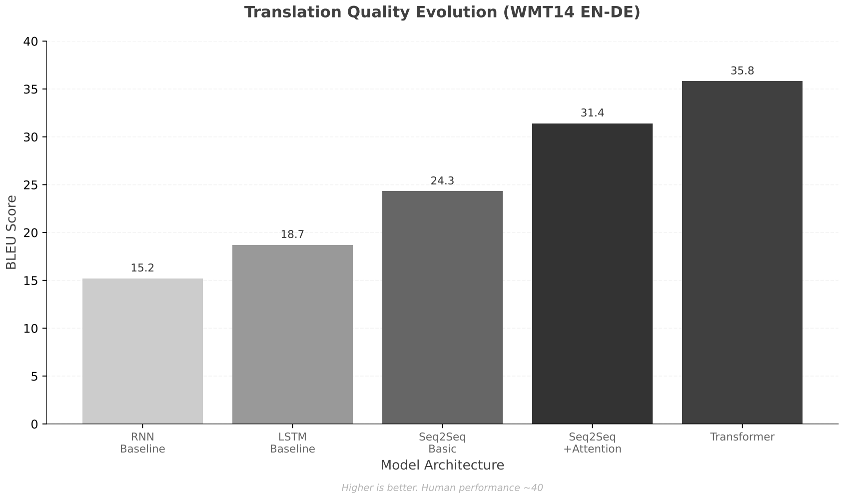
- $s_2 = [0.7, 0.8]$

Your calculations:

1. Scores (dot product):
   - $e_1 = s_2 \cdot h_1 =$ ________
   - $e_2 = s_2 \cdot h_2 =$ ________
   - $e_3 = s_2 \cdot h_3 =$ ________
   - $e_4 = s_2 \cdot h_4 =$ ________
2. Softmax weights:
   - $\alpha_2 =$ ________ (highest!)
3. Context: weighted sum

Hands-On: Computing attention manually builds intuition for the mechanism

Impact of Attention: Dramatic Improvements



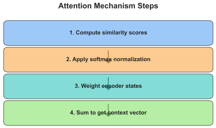
BLEU Score Improvements:

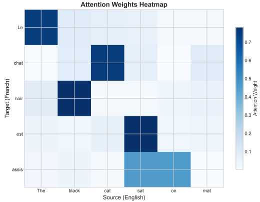
Why It Works:

Part 3 Summary: The Attention Revolution

The Innovation:

- Dynamic context vectors
- Look at all encoder states
- Weighted by relevance
- Different for each word

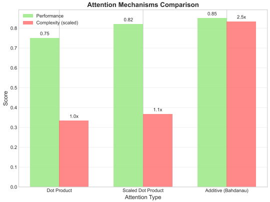


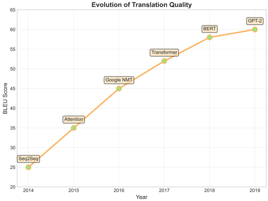


Mathematical Core:

$$\alpha_{ti} = \mathrm{softmax}(\text{score}(s_t, h_i)) \tag{11}$$

$$c_t = \sum_i \alpha_{ti} h_i \tag{12}$$





Impact:
Attention mechanism became foundation of all modern NLP

Historical Significance: Attention paper (2014) revolutionized entire field

Part 4: Implementation & Applications

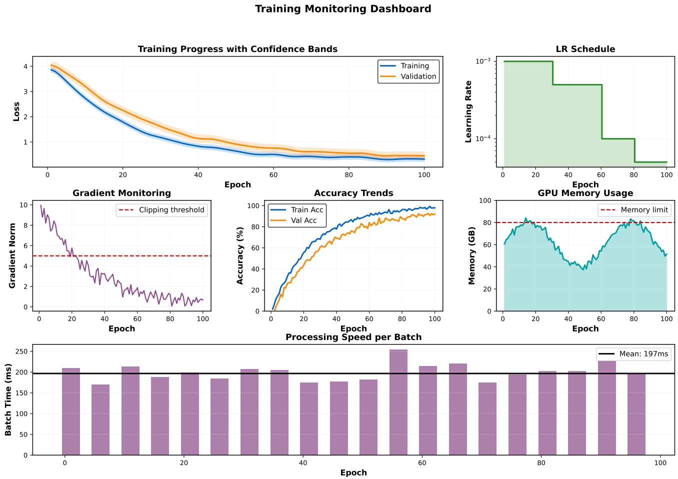
From Research to Production

Complete Seq2Seq with Attention

```python
class AttentionSeq2Seq(nn.Module):
    def __init__(self, src_vocab,
                 tgt_vocab, dim=512):
        super().__init__()

        # Components
        self.encoder = Encoder(
            src_vocab, dim
        )
        self.decoder = DecoderWithAttn(
            tgt_vocab, dim
        )
        self.attention = Attention(dim)

    def forward(self, src, tgt,
                teacher_forcing=0.5):
        # Encode all at once
        enc_out, (h, c) = self.encoder(src)

        batch = src.size(0)
        max_len = tgt.size(1)
        vocab = self.decoder.vocab_size

        # Store outputs
        outputs = torch.zeros(
            batch, max_len, vocab
        )

        # First input
        input = tgt[:, 0]
```

```python
        for t in range(1, max_len):
            # Attention context
            context, weights =
                self.attention(
                    h, enc_out
                )

            # Decode one step
            output, (h, c) =
                self.decoder(
                    input, (h, c),
                    context
                )

            outputs[:, t] = output

            # Teacher forcing
            use_teacher = random.random()
                < teacher_forcing

            if use_teacher:
                input = tgt[:, t]
            else:
                input = output.argmax(1)

        return outputs
```

Training Dynamics: Learning to Translate



Early Training:

- Random attention

Mid Training:

- Alignment emerges

Convergence:

- Sharp attention

Training Best Practices

Hyperparameters That Matter:

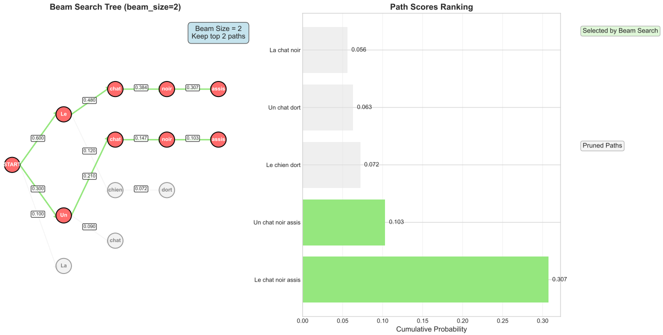
- **Learning rate:** Start 0.001, decay after epoch 10
- **Teacher forcing:** 0.5 $\rightarrow$ 0 over training
- **Gradient clip:** Essential (1.0 works well)
- **Batch size:** 32-64 optimal for GPU
- **Hidden size:** 512 is sweet spot
- **Layers:** 2-3 LSTM layers sufficient

Common Issues & Solutions:

- **Exploding loss:** $\rightarrow$ Reduce learning rate
- **Mode collapse:** $\rightarrow$ Add dropout (0.3)
- **Poor rare words:** $\rightarrow$ Increase min frequency
- **Slow training:** $\rightarrow$ Use GPU, reduce batch size
- **Overfitting:** $\rightarrow$ More data, regularization
- **Underfitting:** $\rightarrow$ Bigger model, longer training

Pro Tip: Start simple, add complexity gradually, monitor validation metrics closely

Beam Search: Better Decoding Strategy



Greedy vs Beam Search:

- Greedy: Pick best at each step
- Beam: Keep top-k hypotheses
- Explore multiple paths
- Better final translations

Beam size trade-off:

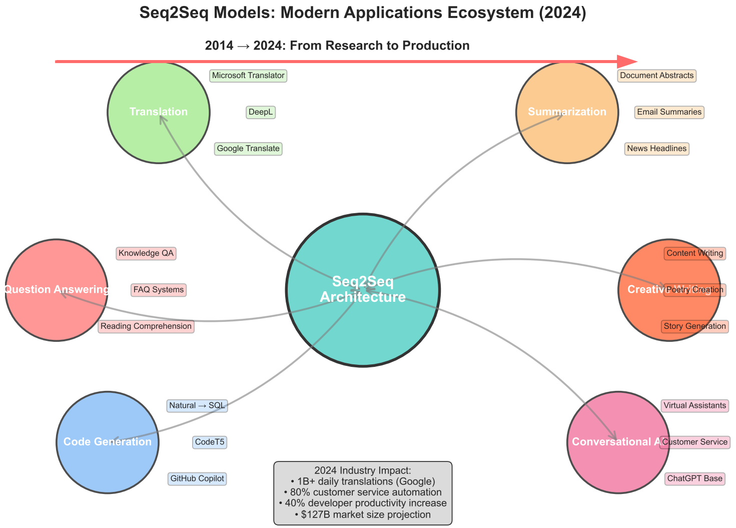
Example with beam=3:

Step 1: "Le", "Un", "Les"
Step 2:

- "Le chat", "Le chien"
- "Un chat", "Les chats"

Step 3: Keep expanding top-3

Final: Pick highest scoring complete hypothesis


Seq2Seq Models: Modern Applications Ecosystem (2024)
2014 → 2024: From Research to Production
Translation
Microsoft Translator
DeepL
Google Translate
Summarization
Document Abstracts
Email Summaries
News Headlines
Question Answering
Knowledge QA
FAQ Systems
Reading Comprehension
Seq2Seq Architecture
Creative Writing
Content Writing
Poetry Creation
Story Generation
Code Generation
Natural → SQL
CodeT5
GitHub Copilot
Conversational AI
Virtual Assistants
Customer Service
ChatGPT Base
2024 Industry Impact:
• 1B+ daily translations (Google)
• 80% customer service automation
• 40% developer productivity increase
• $127B market size projection

Modern Applications: Beyond Translation

Attention Everywhere:

- ChatGPT/Claude (attention-based)
- Image captioning
- Video understanding
- Code generation
- Music composition
- Speech recognition
- Medical diagnosis

Foundation Truth:

Seq2Seq + Attention
=
Modern AI Backbone

Every modern language model
builds on this architecture

Attention mechanism is the foundation of the entire AI revolution

Week 4 Lab: English-French Neural Machine Translation

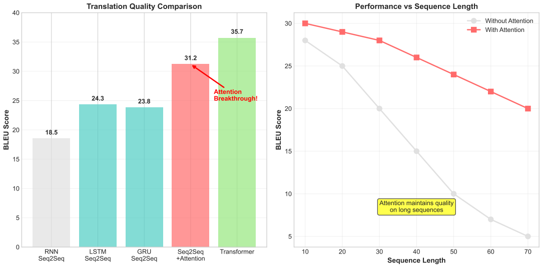
What You'll Build:

1. Load parallel corpus
2. Tokenize and preprocess
3. Implement encoder-decoder
4. Add attention mechanism
5. Train on GPU
6. Visualize attention weights
7. Compare with/without attention

Dataset:

- 10,000 sentence pairs
- English → French
- Average 15 words/sentence

Expected Results:



Bonus Challenges:

- Multi-head attention
- Bidirectional encoder
- Coverage mechanism
- Back-translation

Practical Experience: Implementing attention from scratch solidifies understanding

Interactive Debugging: Common Training Issues

Your model isn't learning. Debug these issues:

Issue 1: Attention all uniform
Symptoms:
- All weights $\approx 1/n$
- Poor translation quality
- Not improving

Your fix: ______________________
Hint: Check score function

Issue 2: Mode collapse
Symptoms:
- Always generates "the the the"
- Loss plateaus high

Common Fixes:

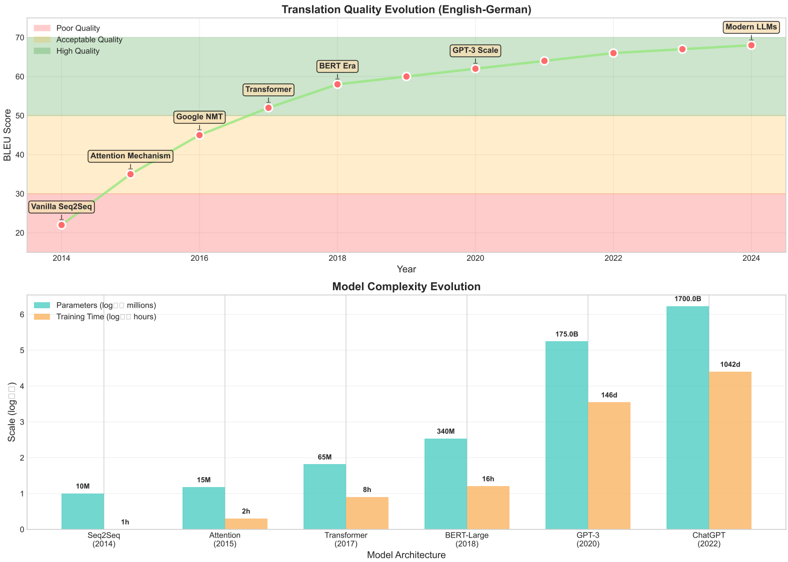
Fix 1: Initialize properly
- Use Xavier initialization
- Scale attention scores
- Add small epsilon to softmax

Fix 2: Teacher forcing
- Start with 100% teacher forcing
- Gradually reduce ratio
- Scheduled sampling

Debug systematically!

Debugging Skills: Most issues come from initialization or training schedule

Performance Evolution Timeline



Performance Metrics by Year

2014: Seq2Seq

- BLEU: 20-25
- Simple, elegant
- Length problems
- 2-3 days training

Impact:

- First neural MT
- Proof of concept
- Beat phrase-based

2015: + Attention

- BLEU: 30-35
- Handles length
- Interpretable
- 4-5 days training

Impact:

- Production ready
- Google adoption
- 40% improvement

2017: Transformer

- BLEU: 40+
- All attention
- Parallel training
- 12 hours training!

Impact:

- New paradigm
- Enables GPT/BERT
- 10x faster training

Key Insight: Each innovation built on the previous - attention was THE breakthrough

Historical Progression: From RNN to attention to transformer architecture

The Bridge to Transformers (Week 5 Preview)

From Seq2Seq+Attention to Transformers:

What We Keep:

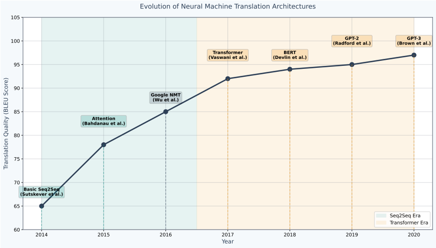
- Attention mechanism
- Query-Key-Value
- Position awareness
- Encoder-decoder structure

What We Remove:

- RNN/LSTM cells
- Sequential processing
- Recurrent connections
- Hidden state passing

What We Add:

- Self-attention
- Multi-head attention
- Position encodings
- Layer normalization
- Parallel processing



Week 4 Complete: From Translation to Attention

Part 1: Challenge

- Translation $\neq$ word replacement
- Need meaning understanding
- Information bottleneck problem

Part 2: Seq2Seq

- Encoder-decoder architecture
- Fixed context vector
- Works but limited by bottleneck

Part 3: Attention

- Dynamic context vectors
- Look at all encoder states
- Massive performance improvement

Part 4: Applications

- Complete implementation
- Beam search decoding
- Powers modern translation
- Foundation for transformers

Key Takeaways:

1. Context vectors compress meaning
2. Attention removes bottleneck
3. Foundation of modern NLP
4. Bridge to transformers

Next Week: Transformers - Attention Without RNNs!

Achievement Unlocked: You understand the foundation of all modern language AI!