Lecture 15: Attention Mechanisms

Week 5, Lecture 1 - From Seq2Seq to Attention

PSYC 51.07: Models of Language and Communication

Winter 2026

Today's Agenda 📋

1. 🤔 **The Context Problem**: Why static embeddings aren't enough
2. 🔄 **Sequence-to-Sequence Models**: The foundation
3. ⚠️ **The Bottleneck Problem**: Why vanilla Seq2Seq struggles
4. ⚡ **Attention Mechanisms**: The breakthrough innovation
5. 🔍 **How Attention Works**: Step-by-step computation
6. 👁️ **Visualizing Attention**: Interpreting the weights

Goal: Understand why and how attention revolutionized NLP

The Context Problem 🤔

Why do we need context-aware models?

Example: The word "bank"

1. "I deposited money at the **bank**" (financial institution)
2. "We sat by the river **bank**" (riverside)
3. "The plane started to **bank** left" (tilt/turn)

Static Embeddings (Word2Vec):

- One vector per word
- Context-independent
- "bank" = [0.2, -0.5, 0.8, ...] always

Context-Aware Models:

Sequence-to-Sequence Models 🔄

The breakthrough for variable-length input/output problems

Applications:

- Machine Translation: English → French
- Summarization: Long text → Short summary
- Question Answering: Question + Context → Answer
- Dialogue Systems: User input → System response

Concrete Example: Machine Translation

Input: "The cat sat on the mat" (6 tokens)

Output: "Le chat s'est assis sur le tapis" (7 tokens)

Different input/output lengths require flexible architecture!

Encoder-Decoder Architecture 🏗️

Two-part architecture: Encode then Decode

Encoder:

- Reads input sequence
- Compresses to fixed-size vector
- Captures semantic meaning

Decoder:

- Starts from context vector
- Generates output sequence
- One token at a time

Worked Example:

The Seq2Seq Bottleneck Problem ⚠️

Challenge: All information compressed into single vector!

The Problem

- Long sequences → information loss
- Fixed-size context vector is a bottleneck
- Early tokens forgotten by the time we reach the end
- Performance degrades with sequence length

Concrete Example: Long Sentence Translation

Input (20 words): "The quick brown fox jumps over the lazy dog while the cat watches from the warm sunny windowsill nearby"

Problem: All 20 words must fit into one 256-dim vector!

- Early words ("The quick brown") get overwritten

Attention Mechanism: The Big Idea ⚡

Instead of compressing everything into one vector...

Let the decoder look at all encoder hidden states!

```
1$h_1$ -> $h_2$ -> $h_3$ -> $h_4$ -> Encoder: -> $s_t$ -> Decoder: -> Input: "Th
```

Key Insight: When generating "chat" (cat), pay more attention to "cat" in the input!

Reference: Bahdanau et al. (2015) - "Neural Machine Translation by Jointly Learning to Align and Translate"

How Attention Works: Step by Step 🔍

Computing attention weights:

1. **Score**: How relevant is each encoder state to current decoder state?
   - $e_{t,i} = \text{score}(s_t, h_i) = s_t^T W_a h_i$
2. **Normalize**: Convert scores to probabilities (softmax)
   - $\alpha_{t,i} = \exp(e_{t,i}) / \sum_j \exp(e_{t,j})$
3. **Context**: Weighted sum of encoder states
   - $c_t = \sum_i \alpha_{t,i} * h_i$
4. **Decode**: Use context vector along with decoder state
   - $s_{t+1} = f(s_t, c_t, y_t)$

Result: Decoder dynamically focuses on different parts of input!

Worked Example: Attention Computation 📊

Translating "I love cats" → "J'aime les chats"

Step 1: Encoder produces hidden states

```
h₁ = [0.2, 0.8]  ("I")
h₂ = [0.9, 0.3]  ("love")
h₃ = [0.4, 0.7]  ("cats")
```

Step 2: When generating "chats", compute scores

- Decoder state $s = [0.5, 0.6]$
- Score with h_1: $s \cdot h_1 = 0.5 \times 0.2 + 0.6 \times 0.8 =$ **0.58**
- Score with h_2: $s \cdot h_2 = 0.5 \times 0.9 + 0.6 \times 0.3 =$ **0.63**
- Score with h_3: $s \cdot h_3 = 0.5 \times 0.4 + 0.6 \times 0.7 =$ **0.62**

Attention Score Functions 📐

Different ways to compute the score

1. Additive (Bahdanau):

```
# score = v^T * tanh(W1*s + W2*h)
score = v @ tanh(W1 @ s + W2 @ h)
```

- More parameters, flexible

2. Multiplicative (Luong):

```
# score = s^T * W * h
score = s @ W @ h
```

- Simpler, faster

Attention Visualization 👁️

Example: English → French translation with attention weights

Input: "The European Economic Area"

Output: "La zone economique europeenne"

```
              The     European  Economic   Area
La           [0.8]     0.1       0.05      0.05
zone         0.05     [0.1]      0.1      [0.75]  ← "zone" = "Area"
economique   0.05      0.1      [0.8]      0.05
europeenne   0.05     [0.8]      0.1       0.05   ← reordering!
```

Observations:

- Diagonal pattern for similar word order
- Model learns alignment automatically!

Benefits of Attention Mechanisms ✅

1. **Solves the Bottleneck Problem**
    - Decoder has access to all encoder states

- No information compression into single vector
- Works well for long sequences

2. **Improves Performance**
    - Better BLEU scores on translation tasks

- Handles long-range dependencies
- More robust to sequence length

3. **Provides Interpretability**
    - Can visualize what the model focuses on

- Helps debug and understand model behavior

Real-World Impact of Attention 🌍

Attention mechanisms revolutionized multiple domains:

Machine Translation:

- Google Translate (2016)
- DeepL
- Facebook translations
- Dramatic quality improvements

Text Summarization:

- News article summarization
- Document understanding
- Email auto-responses

Implementing Attention in PyTorch 💻

Simple attention mechanism implementation

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class BahdanauAttention(nn.Module):
    def __init__(self, hidden_dim):
        super().__init__()
        self.W_dec = nn.Linear(hidden_dim, hidden_dim)
        self.W_enc = nn.Linear(hidden_dim, hidden_dim)
        self.v = nn.Linear(hidden_dim, 1)

    def forward(self, decoder_hidden, encoder_outputs):
        # Compute scores: how relevant is each encoder state?
        dec = self.W_dec(decoder_hidden).unsqueeze(1)  # [batch, 1, hidden]
        enc = self.W_enc(encoder_outputs)              # [batch, seq_len, hidd
        scores = self.v(torch.tanh(dec + enc))         # [batch, seq_len, 1]
```

Implementing Attention in PyTorch 💻

```
        # Weighted sum of encoder outputs
        context = torch.sum(attn_weights * encoder_outputs, dim=1)

        return context, attn_weights.squeeze(-1)
```

...continued

Using the Attention Module 💻

Complete example with sample data

```
# Initialize attention module
attn = BahdanauAttention(hidden_dim=64)

# Sample encoder outputs (3 words, 64-dim hidden state)
encoder_outputs = torch.randn(1, 3, 64)  # [batch=1, seq_len=3, hidden=64]

# Current decoder hidden state
decoder_hidden = torch.randn(1, 64)      # [batch=1, hidden=64]

# Compute attention
context, weights = attn(decoder_hidden, encoder_outputs)

print(f"Context shape: {context.shape}")    # [1, 64]
print(f"Attention weights: {weights}")       # [1, 3] - sums to 1.0!

# Example output:
```

Discussion Questions 💭

1. **Why is attention called "soft alignment"?**
   - How is it different from hard alignment?

- What are the advantages of soft vs. hard?

2. **Computational Cost:**
   - What is the time complexity of attention?

- How does it scale with sequence length?
- When might this be a problem?

3. **Interpretability:**
   - Can we always trust attention weights as explanations?

- What about when attention is uniform across all inputs?

Looking Ahead 🔮

What's Next?

Today we learned:

- The context problem in NLP
- Sequence-to-sequence architecture
- The bottleneck problem
- How attention mechanisms work
- Attention as alignment and interpretation

Next lecture (Lecture 13):

- : Attention within a sequence
- : "Attention is All You Need"

Summary 🎯

Key Takeaways:

1. **Context Matters**
    - Static embeddings can't capture context-dependent meanings

- Need dynamic representations based on context

2. **Seq2Seq Bottleneck**
    - Fixed-size context vector limits performance

- Information loss for long sequences

3. **Attention is the Solution**
    - Dynamic access to all encoder states

- Weighted combination based on relevance

References 📚

Key Papers:

- **Sutskever et al. (2014)** - "Sequence to Sequence Learning with Neural Networks"
- Introduced encoder-decoder architecture
- Foundation for seq2seq models

  \item **Bahdanau et al. (2015)** - "Neural Machine Translation by Jointly Learning to Align and Translate"

  - Introduced additive attention mechanism
- Solved the bottleneck problem

  \item **Luong et al. (2015)** - "Effective Approaches to Attention-based Neural Machine Translation"

Questions? 🙋

Discussion Time

Office Hours Topics:

- Implementing attention from scratch
- Different attention mechanisms
- Debugging attention-based models
- Assignment 4 preparation

Thank you! 🙏

See you next lecture for Transformers!