Sequence-to-Sequence Models

Maps input sequence to output sequence

Different lengths allowed (input ≠ output)

Input Sequence
Variable length
→
Encoder
Processes input
→
Decoder
Generates output
→
Output Sequence
Variable length

🎯 Key Feature

- Handles variable-length sequences
- End-to-end learning

💼 Applications

- Machine Translation
- Text Summarization
- Conversation Systems

Bottleneck Challenge

Entire input compressed to fixed vector → Attention mechanism addresses this

🔄 Step-by-Step Process

1 **Input Encoding**

Encoder RNN processes input sequence token by token, updating hidden state

2 **Context Vector**

Final encoder hidden state becomes context vector (fixed representation)

3 **Decoder Initialization**

Context vector initializes decoder RNN's hidden state

4 **Sequential Generation**

Decoder generates output tokens one at a time, using previous output as next input

5 **Termination**

Process continues until special end-of-sequence token is generated

🔢 Numerical Example with 3D Vectors

Weight Matrices (Encoder)

W (input weight, 3×3): `[[0.5, 0.2, 0.3], [0.1, 0.6, 0.4], [0.3, 0.2, 0.7]]`

U (hidden weight, 3×3): `[[0.4, 0.3, 0.2], [0.2, 0.5, 0.3], [0.3, 0.2, 0.6]]`

Input Sequence

x_1 = `[0.5, 0.3, 0.8]` → x_2 = `[0.2, 0.7, 0.4]`

⚠

Encoder Step 1

h_0 = `[0, 0, 0]` (initial hidden state)

$\mathbf{W} \cdot \mathbf{x_1}$ = [[0.5, 0.2, 0.3], [0.1, 0.6, 0.4], [0.3, 0.2, 0.7]] · [0.5, 0.3, 0.8] = `[0.55, 0.47, 0.77]`

$\mathbf{U} \cdot \mathbf{h_0}$ = `[0, 0, 0]`

h_1 = tanh([0.55, 0.47, 0.77]) ≈ `[0.50, 0.44, 0.65]`

Encoder Step 2

$\mathbf{W \cdot x_2}$ = [[0.5, 0.2, 0.3], [0.1, 0.6, 0.4], [0.3, 0.2, 0.7]] · [0.2, 0.7, 0.4] = `[0.38, 0.58, 0.48]`

$\mathbf{U \cdot h_1}$ = [[0.4, 0.3, 0.2], [0.2, 0.5, 0.3], [0.3, 0.2, 0.6]] · [0.50, 0.44, 0.65] = `[0.46, 0.42, 0.61]`

h_2 = tanh([0.38 + 0.46, 0.58 + 0.42, 0.48 + 0.61]) = tanh([0.84, 1.00, 1.09]) ≈ `[0.69, 0.76, 0.80]`

Context Vector c = h_2 = [0.69, 0.76, 0.80]

Weight Matrices (Decoder)

W' (input weight, 3×3): `[[0.6, 0.3, 0.2], [0.2, 0.5, 0.4], [0.4, 0.3, 0.5]]`

U' (hidden weight, 3×3): `[[0.5, 0.2, 0.3], [0.3, 0.6, 0.2], [0.2, 0.3, 0.7]]`

V (output weight, vocab_size×3): maps hidden to output vocabulary

Decoder Step 1

s_0 = `[0.69, 0.76, 0.80]` (initialized with context)

$\mathbf{V \cdot s_0}$ → softmax → y_1 (output token, e.g., word embedding)

Assume y_1 = `[1, 0, 0]` (one-hot encoded)

s_1 = tanh(W'·y_1 + U'·s_0) = tanh([0.6, 0.2, 0.4] + [0.82, 0.84, 0.91]) ≈ `[0.77, 0.72, 0.80]`

Decoder Step 2

$\mathbf{V \cdot s_1}$ → softmax → y_2 (next output token)

s_2 = tanh(W'·y_2 + U'·s_1) → continue...

Process continues until **<END>** token is generated