Course Materials for GEN-AI

Northeastern University

These materials have been prepared and sourced for the course **GEN-AI** at Northeastern University. Every effort has been made to provide proper citations and credit for all referenced works.

If you believe any material has been inadequately cited or requires correction, please contact me at:

Instructor: Ramin Mohammadi
r.mohammadi@northeastern.edu

Thank you for your understanding and collaboration.

Content sourced from multiple references - see Reference section for details

Attention Mechanisim

Introduction

The attention mechanism was first introduced in 2014 by Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio in their paper titled *"Neural Machine Translation by Jointly Learning to Align and Translate"*. This paper introduced what is now known as **Bahdanau attention**, or **soft attention**, and was applied to neural machine translation (NMT).

In this approach, instead of using a fixed-length context vector for the entire sequence, the model generates a dynamic context vector at each decoding step, allowing it to "attend" to different parts of the input sequence based on relevance. This was a significant innovation in sequence modeling and became the foundation for subsequent developments in attention mechanisms, including the Transformer model.

Motivation for Attention

To understand the need for Attention, consider the following passage:

> *"The restaurant refused to serve me a ham sandwich because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambiance was just as good as the food and service."*

This example highlights several challenges in natural language processing:

- **Contextual Understanding:** Words like "it" and "their" refer to "restaurant," requiring models to connect words across long distances.
- **Ambiguity:** "it" could refer to either "restaurant" or "ham sandwich."
- **Flexible Connections:** The model needs to create connections between words based on context, not just position.

Traditional models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks struggle with these challenges, especially for longer sequences. Transformer models address these issues through self-attention mechanisms, allowing for efficient processing of variable-length inputs and capturing long-range dependencies in text.

Sequence-to-Sequence Models

Before delving into transformers, it's crucial to understand Sequence-to-Sequence (Seq2Seq) models, which form the foundation for many language tasks.

Architecture

Seq2Seq models consist of two main components:

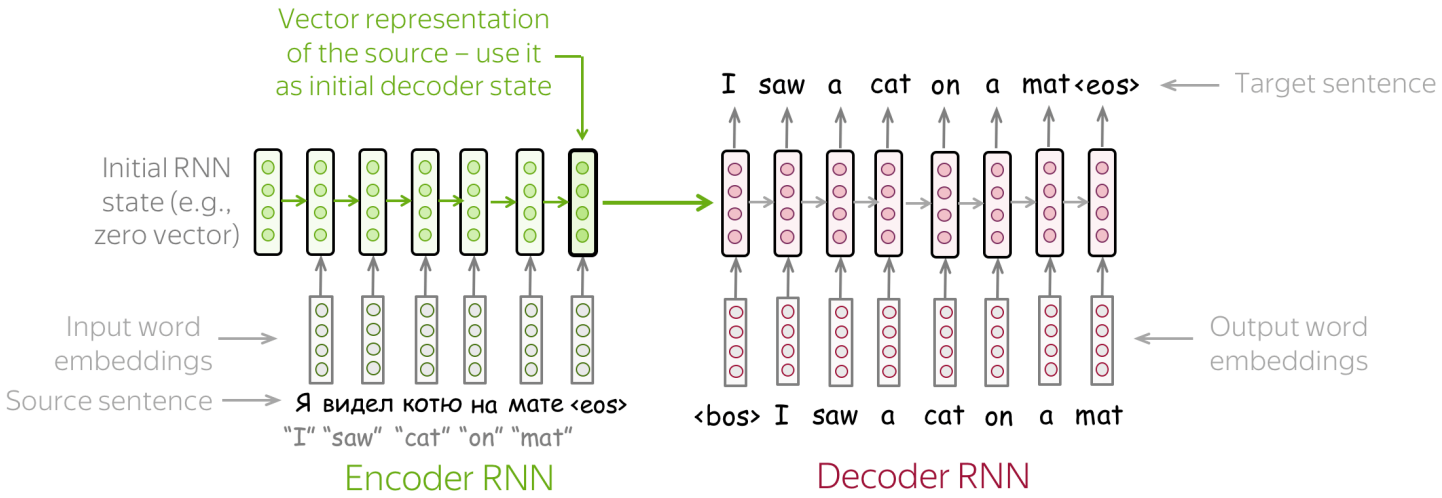

Figure 1: Example of how seq2seq Encoder and decoder model translates language

1. **Encoder:** Processes the input sequence and compresses it into a fixed-length context vector.
2. **Decoder:** Generates the output sequence based on the context vector and previously generated tokens.

Mathematically:

$$\text{context} = f_{\text{encoder}}(x_1, x_2, ..., x_n) \tag{1}$$

$$y_t = f_{\text{decoder}}(\text{context}, y_{t-1}, h_{t-1}) \tag{2}$$

Where x_i are input tokens, y_t is the output token at time t, and h_{t-1} is the hidden state of the decoder at the previous time step.

Decoding Process

The decoder operates as follows:

1. It receives the context vector, which is a representation of the entire input sequence.
2. At each time step t, it takes as input:
   - The context vector
   - The previously generated token y_{t-1}
   - Its own hidden state h_{t-1}
3. It produces a new token y_t based on these inputs.
4. The newly generated token becomes the input for the next time step.

This process can be expressed as:

$$P(y_1, ..., y_m | x_1, ..., x_n) = \prod_{t=1}^{m} P(y_t | \text{context}, y_1, ..., y_{t-1}) \tag{3}$$

Challenges of Seq2Seq Models

Despite their success, Seq2Seq models face several critical challenges:

1. **Long-Range Dependencies:** Due to the sequential nature of RNNs, words that are close together in a sequence strongly influence each other's meaning, a phenomenon known as **linear locality**. While this locality can be helpful for capturing nearby relationships, it poses a challenge for longer sequences. Fixed-length context vectors in RNNs often struggle to retain essential information from the beginning of a long input, resulting in the loss of important context and weakening the model's ability to handle long-range dependencies (see Figure 2).

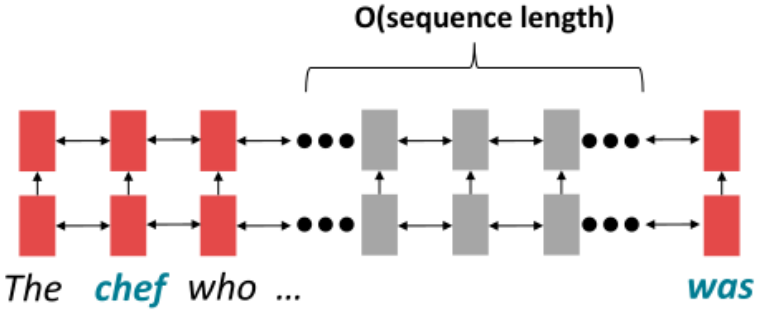

Figure 2: Long Range Dependency

2. **Limited Parallelization:** Modern GPUs excel at performing numerous simple, independent operations in parallel. For instance, they can efficiently compute matrix multiplications like $A \in \mathbb{R}^{n \times k}$ and $B \in \mathbb{R}^{k \times d}$ to obtain $AB \in \mathbb{R}^{n \times d}$, as these operations largely do not depend on each other's output.

   However, in an RNN, each hidden state depends on the previous one, creating a chain of dependencies. For example, to compute

$$h_2 = \sigma(Wh_1 + Ux_2), \tag{5}$$

   we must know the value of h_1. This sequential dependency means that computing h_3 requires knowing h_2, which in turn depends on h_1, and so forth. This process, shown visually in Figure 3, limits the degree of parallelism possible on a GPU. As the sequence length increases, these serial dependencies constrain the ability to speed up computation, as each step must be processed in order.

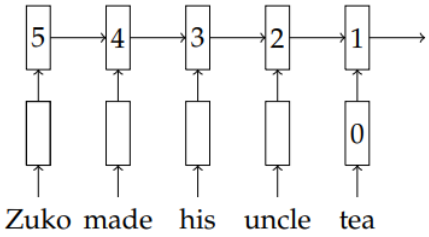

Figure 3: An RNN unrolled in time. The rectangles represent intermediate states of the RNN (e.g., the first row is the embedding layer, and the second row is the RNN hidden state at each time step). The numbers within the rectangles denote the number of sequential operations needed before each intermediate state can be computed.

   As GPUs (and later, other accelerators like Tensor Processing Units (TPUs)) became more powerful, researchers wanted to fully leverage their parallel processing capabilities. However, the time-dependent nature of RNNs makes it challenging to exploit this parallelism effectively.

3. **Vanishing Gradients:** When processing long sequences, RNNs are prone to the vanishing gradient problem during backpropagation, making it difficult for the model to learn long-term dependencies. Gradients tend to shrink as they propagate backward through many layers, reducing the model's ability to capture and retain information from earlier time steps in long sequences.

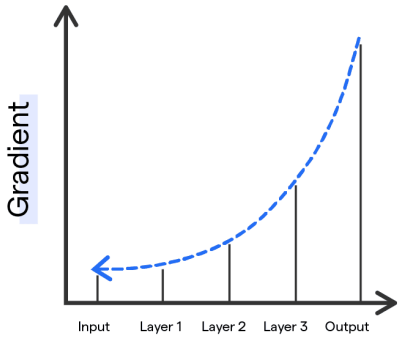

Figure 4: Gradient vanishes as we back-propagate though the network

Improvements

Several techniques have been developed to address these challenges:

- **LSTM and GRU:** These gated recurrent units help mitigate the vanishing gradient problem by allowing information to flow more easily through the network.
- **Attention mechanisms:** Allow the model to focus on different parts of the input sequence when generating each output token, effectively addressing the information bottleneck of fixed-length context vectors.
- **Transformer architecture:** Replaces the recurrent structure with self-attention, allowing for better parallelization and improved handling of long-range dependencies.

Attention Mechanisms

Attention mechanisms were introduced to address the limitations of traditional Seq2Seq models, particularly the challenge of capturing long-range dependencies.

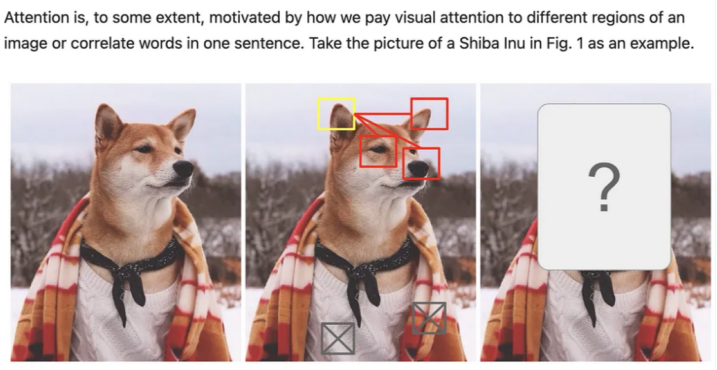

Figure 5: Attention Motivation

Concept of Attention

Figure 6: One word "attend" to other words in the same sentence differently

Attention can be thought of as a **weighted average, directing focus to important factors when processing data**. In the context of NLP, it allows the model to weigh the importance of different words in the input sequence when generating each word in the output sequence.

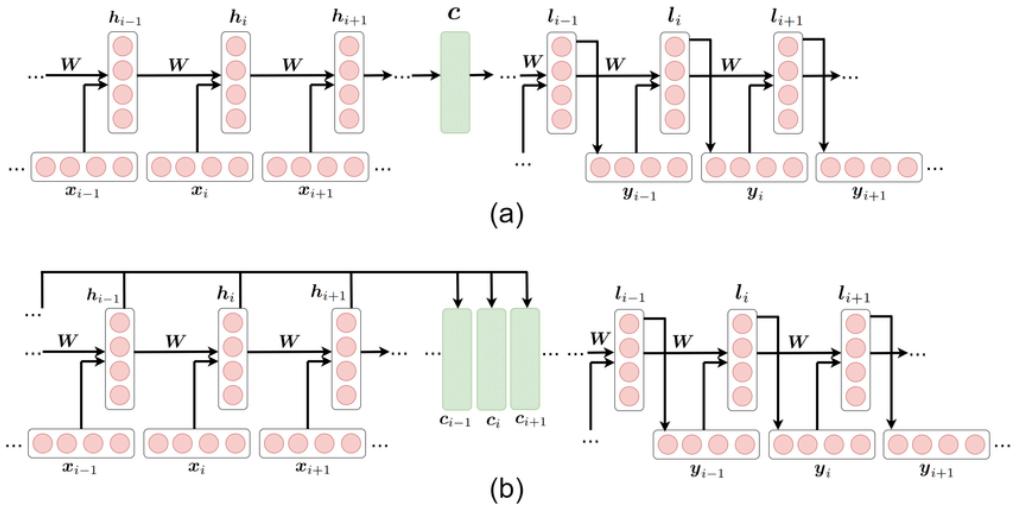

Figure 7: (a) - General seq2seq model. (b) - seq2seq model with attention mechanisim

General Seq2Seq Models

As discussed earlier, the decoder in a Seq2Seq model predicts the next token based on the previous token and the context vector C (Figure 5 - a). The context vector C is produced by the encoder after it processes the entire input sequence, capturing essential information from the full sequence.

$$p(y_i|y_1, y_2, ..., y_{i-1}) = g(y_{i-1}, l_i, c) \tag{4}$$

Attention in Seq2Seq Models

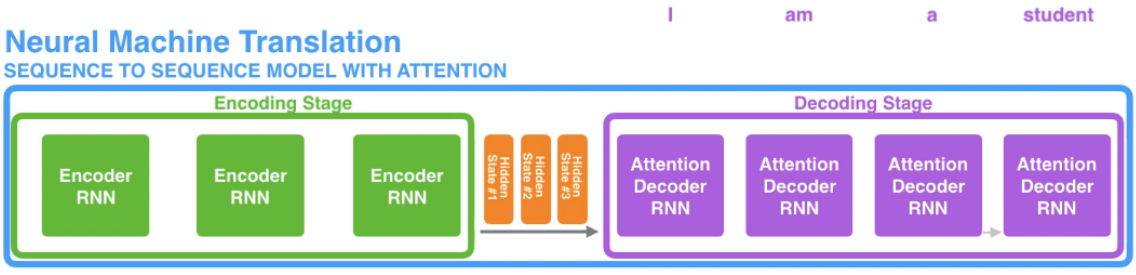

Figure 8: Seq2Seq model with attention

In a Seq2Seq model with attention, the encoder generates a context vector at each time step, encapsulating the learned context up to that point (Figure 5 - b). This enables the model to focus on different parts of the input sequence when predicting each output token. The conditional probability of predicting the next token $\hat{y}_t$ given the previous tokens and the context vector at time step t is:

$$p(y_i|y_1, y_2, ..., y_{i-1}) = g(y_{i-1}, l_i, c_i) \tag{5}$$

In an attention-based Seq2Seq model, the context vector at each time step is **dynamically computed based on attention scores**, allowing the model to **"attend" to different encoder states relevant to the current decoding step**. The context vector c_i at time step i is computed using attention weights as follows:

$$c_i = a_1 h_1 + a_2 h_2 + ... + a_T h_T \tag{6}$$

where:

- a_j is the attention weight for the encoder hidden state at time j, with $\sum_{j=1}^{T} a_j = 1$
- h_j is the hidden state of the encoder at time j

$$c_i = \sum_{j=1}^{k} a_{ij} h_j \tag{7}$$

Computing Attention Weights

Figure 9: Seq2Seq model with attention

The attention weights are computed using a similarity function and softmax, where each attention weight a_{ij} represents the importance of the encoder hidden state h_j for generating the decoder hidden state L_i:

$$S_{ij} = \text{similarity}(L_{i-1}, h_j) \tag{8}$$

$$a_{ij} = \frac{e^{S_{ij}}}{\sum_{j=1}^{k} e^{S_{ij}}} \tag{9}$$

where:

- S_{ij} is the similarity score between the decoder's previous hidden state L_{i-1} and the encoder hidden state h_j. This score indicates the relevance of h_j for predicting y_i, which depends on L_{i-1}.

- a_{ij} is the normalized attention weight (using Softmax function) for the encoder hidden state h_j at decoding step i, with $\sum_{j=1}^{k} a_{ij} = 1$.

The similarity function could be a simple dot product, a neural network, or any other function that captures the relationship between the decoder state and encoder hidden states.

Attention Weights

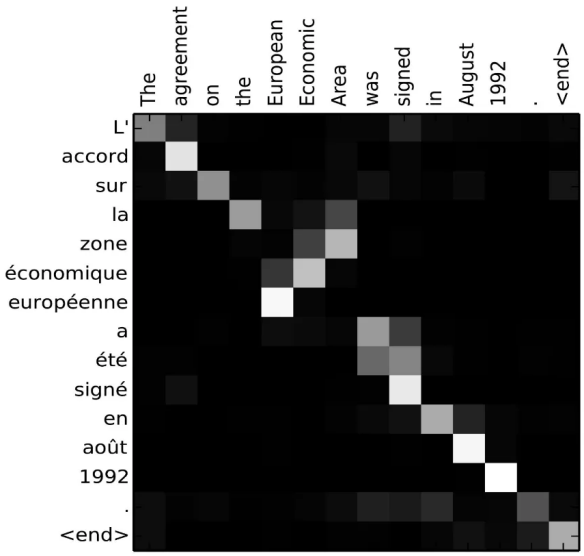

Figure 10: Attention weights in NMT

Detailed Attention Process in Seq2Seq

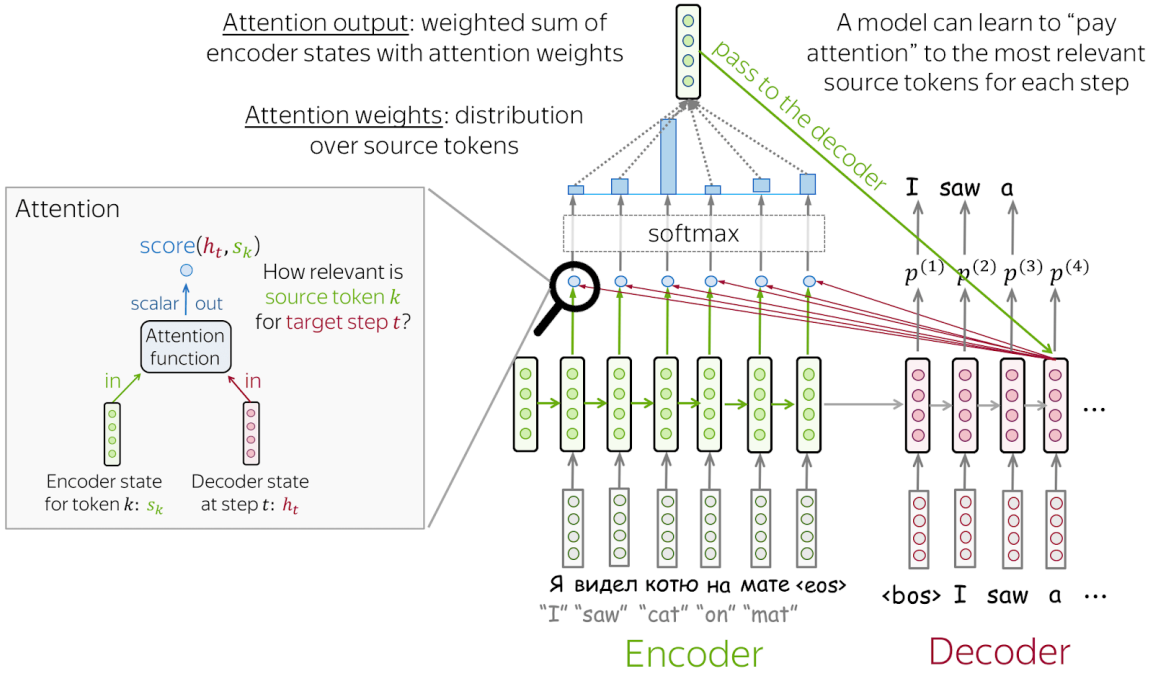

Figure 11: Attention in translation

Encoding Phase

The encoder processes the input sequence $\mathbf{X} = (x_1, x_2, ..., x_T)$ to generate hidden states $\mathbf{H} = (h_1, h_2, ..., h_T)$. Each hidden state h_j represents the encoding of the input up to and including the j-th token.

Decoding Phase with Attention

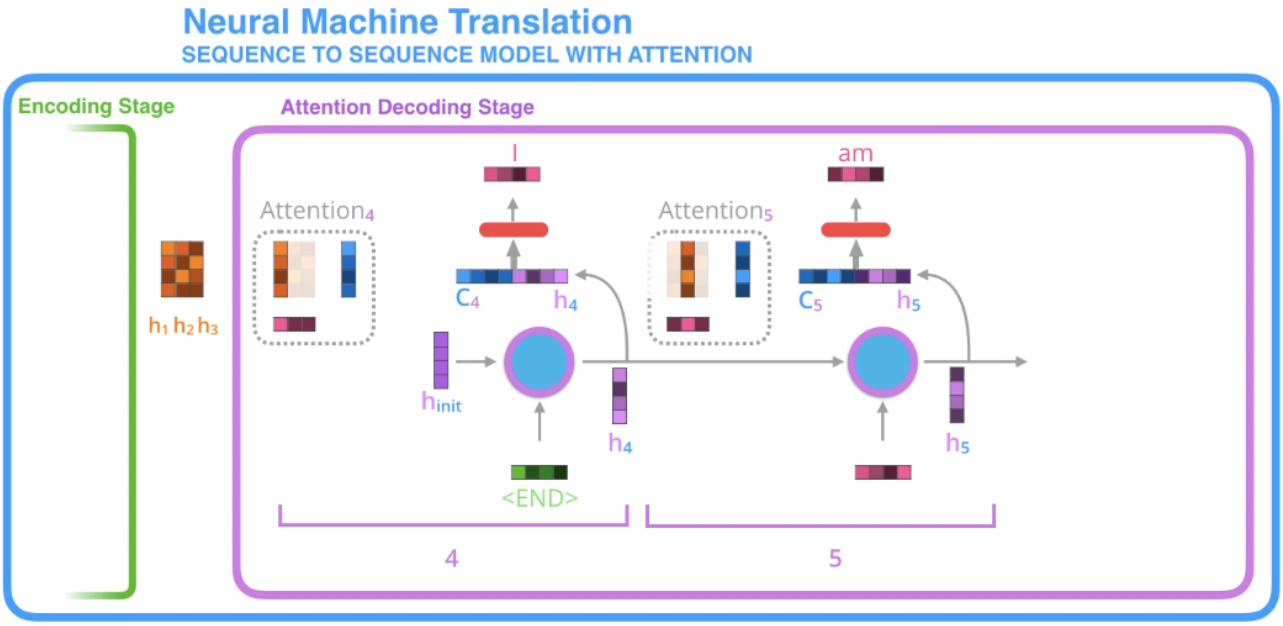

Figure 12: Decoder stage with attention mechanism

For each time step t:

1. Compute attention weights:
$$\alpha_{tj} = \frac{\exp(\text{score}(s_t, h_j))}{\sum_{j=1}^{T} \exp(\text{score}(s_t, h_j))}$$
This step calculates how much attention should be paid to each input token when generating the current output token.

2. Compute context vector:
$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$$
The context vector is a weighted sum of all encoder hidden states, with weights determined by the attention mechanism.

3. Update decoder hidden state:
$$s_t = \text{Decoder}(y_{t-1}, s_{t-1}, c_t)$$
The decoder uses the previous output token, its previous hidden state, and the context vector to update its current hidden state.

4. Generate output token:
$$y_t = \text{Output}(s_t)$$
The current hidden state is used to generate the output token for this time step.

This process allows the model to focus on different parts of the input sequence at each decoding step, effectively addressing the limitations of fixed-length context vectors.

References

[1] Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

[2] John Hewitt. "Self-Attention & Transformers.".

[3] Alammar, J (2018). The Illustrated Transformer .