Revision: Attention Mechanism

Minor in AI - IIT ROPAR

29th April, 2025

1 Attention Mechanism

Why Attention?

In traditional sequence-to-sequence (seq2seq) models, the encoder compresses the entire source sentence into a fixed-length context vector. This becomes problematic for longer sentences due to limited memory capacity.

The attention mechanism overcomes this by allowing the decoder to dynamically focus on different parts of the source sentence during generation.

How Does Attention Work?

Consider translating *"The cat is on the mat"* into French. When generating the French word *"chat"*, the decoder should focus more on the word *"cat"* in the source sentence.

The attention mechanism assigns weights to each word in the source sentence based on its relevance to the current word being generated.

Core Steps:

1. Compute similarity scores between the decoder's current hidden state (query) and each encoder hidden state (keys).
2. Apply softmax to normalize scores into attention weights (probabilities).
3. Compute the weighted sum of encoder hidden states (values), forming a context vector.
4. Use this context vector to help the decoder generate the next word.

2 Self-Attention?

Self-attention, also known as intra-attention, is a mechanism that allows a model to relate different positions of a single input sequence in order to compute a context-aware representation of that sequence. It was popularized by the Transformer architecture, which avoids recurrence and processes input tokens in parallel. This mechanism enables each word to attend to other words in the same sentence, capturing both local and global dependencies.

Consider the sentence: *"The cat sat on the mat."* To understand the word "sat," it is helpful for the model to know what the subject ("cat") is. Self-attention allows the representation of "sat" to incorporate information from "cat," regardless of their positions in the sequence.

Why Use Self-Attention?

Traditional recurrent models (RNNs, LSTMs) encode sequences in a step-by-step manner. This sequential processing is inherently slow and makes it difficult to capture long-range dependencies due to vanishing gradients or information bottlenecks. Self-attention addresses this by allowing every token to directly attend to every other token, enabling full connectivity and parallel computation.

Step-by-Step Computation

1. Input Representation

Let us consider a sequence of n tokens as the input to the self-attention mechanism. Each token is represented by a d-dimensional embedding vector. We stack these token embeddings row-wise to form the input matrix:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} \in \mathbb{R}^{n \times d}$$

Here:

- n is the sequence length (number of tokens).
- d is the dimensionality of each token embedding.
- $\mathbf{x}_i \in \mathbb{R}^d$ is the embedding of the i^{th} token.

2. Linear Projections: Queries, Keys, and Values

In self-attention, we compute three linear transformations of the input sequence:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V$$

where:

- $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d \times d_k}$ are learned weight matrices.
- $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d_k}$ are the resulting matrices for queries, keys, and values respectively.

Each row of $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ corresponds to a transformed version of each token embedding.

3. Computing Attention Scores (Raw Compatibility)

The attention mechanism is fundamentally about measuring the compatibility (or similarity) between a token's query vector and all other tokens' key vectors. For a given token i, we compute:

$$\text{score}(i, j) = \mathbf{q}_i \cdot \mathbf{k}_j = \langle \mathbf{q}_i, \mathbf{k}_j \rangle$$

This is a dot product between the i^{th} row of $\mathbf{Q}$ and the j^{th} row of $\mathbf{K}$. To compute all pairwise scores for the full sequence, we perform a matrix multiplication:

$$\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{n \times n}$$

Here, $\mathbf{S}_{ij}$ is the similarity between token i (as a query) and token j (as a key). Each row i in $\mathbf{S}$ contains scores indicating how much the i^{th} token should attend to all tokens $j = 1, \ldots, n$.

4. Scaling the Scores

To prevent the dot products from growing too large as d_k increases—which would push the softmax into regions with very small gradients—we scale the raw scores by the square root of the key/query dimension:

$$\mathbf{S}_{\text{scaled}} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \in \mathbb{R}^{n \times n}$$

This ensures that the magnitude of the dot products remains approximately in the same range regardless of d_k.

5. Softmax Normalization (Attention Weights)

To convert the raw scores into a valid probability distribution over the sequence (i.e., weights that sum to 1), we apply the softmax function row-wise:

$$\mathbf{A}_{ij} = \frac{\exp(\mathbf{S}_{ij}/\sqrt{d_k})}{\sum_{k=1}^{n} \exp(\mathbf{S}_{ik}/\sqrt{d_k})} \quad \text{for } i = 1, \ldots, n$$

This gives us the attention matrix:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right) \in \mathbb{R}^{n \times n}$$

Each row $\mathbf{A}_i$ contains attention weights indicating how much token i should focus on each other token j in the sequence.

6. Computing the Final Output (Contextual Representation)

Using the attention weights in $\mathbf{A}$, we compute a new representation for each token by taking a weighted average over the value vectors:

$$\mathbf{Z} = \mathbf{A}\mathbf{V} \in \mathbb{R}^{n \times d_k}$$

Here:

- Row $\mathbf{z}_i$ of $\mathbf{Z}$ is the new representation of the i^{th} token.
- It is computed as a linear combination of the value vectors, weighted by how much attention token i pays to all other tokens.

Multi-Head Self-Attention

In practice, Transformers use multiple self-attention heads. Each head has its own set of projections:

$$\text{head}_i = \text{SelfAttention}(\mathbf{X}\mathbf{W}_i^Q, \mathbf{X}\mathbf{W}_i^K, \mathbf{X}\mathbf{W}_i^V)$$

These heads operate in parallel, and their outputs are concatenated:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O$$

This allows the model to capture various types of linguistic relationships, such as co-reference, syntax, and positionally distant dependencies.

Example Intuition

For the sentence:

"The animal didn't cross the street because it was too tired."

The model must decide what "it" refers to. Through self-attention, the representation of "it" can attend more heavily to "animal," helping the model resolve this ambiguity.

3 BLEU Score: Bilingual Evaluation Understudy

Introduction

The BLEU score, which stands for **Bilingual Evaluation Understudy**, is one of the most influential and widely used automatic metrics for evaluating machine translation quality. Originally introduced by researchers at IBM in 2002, BLEU provides a quantitative way to evaluate how closely a machine-generated translation resembles a high-quality human translation. The idea is to mimic human judgment to some extent, using a simple yet effective statistical method based on surface-level textual overlap.

Basic Intuition Behind BLEU

Unlike humans, BLEU does not attempt to understand or interpret meaning. Instead, it evaluates translations by calculating the degree of overlap between the candidate (machine-generated) translation and one or more reference (human-generated) translations. Specifically, it computes how many n-grams from the candidate appear in the reference. An n-gram is simply a sequence of n contiguous words. For instance:

- Unigrams ($n = 1$): "I", "love", "cats"
- Bigrams ($n = 2$): "I love", "love cats"
- Trigrams ($n = 3$): "I love cats"

This comparison is done for multiple values of n (usually from 1 to 4), and the results are combined into a single BLEU score that reflects both precision and adequacy of the translation.

Modified n-gram Precision

The BLEU score uses a modified form of n-gram precision. Precision, in this context, refers to the proportion of n-grams in the candidate translation that also appear in the reference translations.

Let us define:

$$\text{Modified Precision} = \frac{\sum_{\text{ngram} \in \text{Candidate}} \min(\text{Count}_{\text{Cand}}(\text{ngram}), \text{MaxRefCount}(\text{ngram}))}{\sum_{\text{ngram} \in \text{Candidate}} \text{Count}_{\text{Cand}}(\text{ngram})}$$

This formula incorporates **clipping**, which prevents the system from inflating the score by repeating a valid n-gram many times. For example, if the reference has the word "the" once, and the candidate has it three times, only one occurrence is considered correct. This mechanism penalizes over-generation and encourages more accurate translation.

Brevity Penalty (BP)

One of the main shortcomings of raw precision is that it favors shorter sentences. A very short sentence might match some unigrams from the reference and yield high precision simply because there are fewer total n-grams to consider. To counteract this, BLEU incorporates a **brevity penalty** (BP), which penalizes candidates that are significantly shorter than the references.

The brevity penalty is computed as:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

where:

- c is the total length of the candidate translation
- r is the total length of the reference translation

This formula ensures that as the candidate translation becomes shorter relative to the reference, the BLEU score is penalized exponentially. However, if the candidate is longer than the reference, no penalty is applied.

Final BLEU Score Formula

The final BLEU score combines the brevity penalty with the geometric mean of the modified n-gram precisions across different orders. It is computed as follows:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

where:

- p_n is the modified precision for n-grams of size n

- w_n is the weight assigned to each p_n (commonly uniform: $w_n = \frac{1}{N}$ for $N = 4$)

The geometric mean ensures that a low score in any n-gram level significantly affects the overall BLEU score. It favors translations that maintain consistency across unigrams, bigrams, trigrams, and four-grams.

Worked Example

Consider the following scenario:

- **Reference**: "The cat is on the mat"
- **Candidate**: "The cat is on mat"

Let us compute the BLEU score step-by-step.

1. **Unigram Precision**: All candidate words except "the" (second occurrence) match with the reference. So, 5 matched unigrams out of 5.

$$p_1 = \frac{5}{5} = 1.0$$

2. **Bigram Precision**: Candidate bigrams: "The cat", "cat is", "is on", "on mat". Out of these, only "on mat" does not match the reference bigram "on the". So, 3 out of 4 bigrams match.

$$p_2 = \frac{3}{4} = 0.75$$

3. **Trigram Precision**: Fewer matches due to the missing "the". Suppose 1 match out of 3.

$$p_3 = \frac{1}{3} \approx 0.33$$

4. **Length Penalty**: Candidate length $c = 5$, reference length $r = 6$.

$$\text{BP} = e^{(1-6/5)} = e^{-0.2} \approx 0.818$$

5. **Final Score** (assuming $N = 3$ and uniform weights $w_n = \frac{1}{3}$):

$$\text{BLEU} = 0.818 \cdot \exp\left(\frac{1}{3}\log(1.0) + \frac{1}{3}\log(0.75) + \frac{1}{3}\log(0.33)\right)$$

$$= 0.818 \cdot \exp\left(0 + (-0.288) + (-1.108)/3\right) = 0.818 \cdot \exp(-0.465) \approx 0.818 \cdot 0.628 \approx 0.514$$

So, the BLEU score is approximately 0.514 for this candidate.

Strengths and Weaknesses of BLEU

Strengths:

- BLEU is fully automatic and reproducible. It does not rely on subjective human judgments.
- It is fast and scalable to large corpora.
- It is language-agnostic and can be applied to any language pair.
- BLEU correlates reasonably well with human judgments at the corpus level.

Weaknesses:

- BLEU does not consider synonyms or paraphrases. If the candidate uses a different word with the same meaning, BLEU may penalize it unfairly.
- It is sensitive to small changes in word order, even if those changes are acceptable.
- Sentence-level BLEU scores are unreliable due to lack of statistical robustness. BLEU is more meaningful at the corpus level.
- BLEU assumes all n-gram matches are equally important, which may not reflect linguistic priorities.

4 Transformers

Introduction

Transformers have revolutionized the field of machine translation (MT) by eliminating recurrence in Recurrent Neural Networks (RNNs) and enabling the parallel processing of tokens using a self-attention mechanism. This design allows the model to capture long-range dependencies between words more efficiently and with greater scalability, making it the backbone of modern machine translation models.

Tokenization and Embeddings

Tokenization is the first step in processing input text. In many modern NLP models, tokenization is performed using methods like *Byte Pair Encoding* (BPE), which splits text into subword units. These subwords are then mapped to dense vector representations, which serve as input to the Transformer model.

Let the tokenized input text be represented by:

$$\mathbf{X} = [x_1, x_2, \ldots, x_n]$$

where x_i is the i-th token.

The embeddings are represented by the matrix $\mathbf{E}$, where each row is a dense vector corresponding to a token in the vocabulary. The vocabulary size is denoted by V, and the embedding dimension by d:

$$\mathbf{E} \in \mathbb{R}^{V \times d}$$

For a given sequence of tokens $\mathbf{T}$ (of size n), the input embeddings $\mathbf{X}$ are obtained by multiplying the embedding matrix $\mathbf{E}$ with the token index matrix $\mathbf{T}$:

$$\mathbf{X} = \mathbf{E} \cdot \mathbf{T}$$

where $\mathbf{T}$ is a vector of token indices in the vocabulary.

Positional Encoding

Transformers do not have any inherent understanding of the sequence order due to their parallel processing nature. To address this, position information is added to the token embeddings via *positional encoding*. This encoding uses sinusoidal functions to inject position-dependent information into each token's representation.

For a given position pos and dimension i, the positional encoding is defined as follows:

$$\text{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$\text{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

Where:

- pos is the position of the token in the sequence,
- i is the dimension of the positional encoding,
- d is the embedding dimension.

The values of $\text{PE}(pos, 2i)$ and $\text{PE}(pos, 2i+1)$ provide distinct signals that allow the model to differentiate between token positions within a sequence.

The final input representation $\mathbf{X}$ for a given token sequence is the sum of the token embeddings and positional encodings:

$$\mathbf{X}_{\text{final}} = \mathbf{X} + \text{PE}$$

Self-Attention Mechanism

The *self-attention mechanism* allows the model to weigh the importance of different tokens in the input sequence relative to each other, enabling it to capture dependencies regardless of distance in the sequence. Self-attention computes a weighted sum of the input tokens, where the weights are determined by the similarity between the token representations.

For each token in the sequence, the following steps are performed:

- Compute *queries*, *keys*, and *values* from the input tokens:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V$$

  where $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are weight matrices for queries, keys, and values, respectively.

- Compute the *attention scores* by taking the dot product between the query and key matrices, scaling by the square root of the dimension d, and applying a softmax function:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V}$$

  where the softmax function normalizes the attention scores across all tokens.

This process allows the model to focus on relevant tokens and ignore irrelevant ones when processing each token in the sequence.

Multi-Head Self-Attention

To capture different types of relationships between tokens, *multi-head attention* is employed. Multiple attention heads are used to compute attention in parallel, and the results are concatenated and passed through a linear transformation.

The multi-head self-attention mechanism can be expressed as:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}_O$$

where each head computes the attention for a different subspace of the queries, keys, and values, and h is the number of attention heads. The resulting concatenated heads are transformed by a learned weight matrix $\mathbf{W}_O$.

Feedforward Layers

After the multi-head attention layer, each token passes through a *position-wise feedforward network* (FFN). This network consists of two linear transformations with a ReLU activation function in between. The feedforward network is applied independently to each token in the sequence.

The feedforward network is defined as:

$$\text{FFN}(x) = \max(0, x\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

where:

- $\mathbf{W}_1$ and $\mathbf{W}_2$ are weight matrices,
- $\mathbf{b}_1$ and $\mathbf{b}_2$ are bias terms,
- $\max(0, x)$ denotes the ReLU activation function.

This feedforward layer is designed to allow the model to learn complex transformations of the input sequence.

Residual Connections and Layer Normalization

To facilitate training and avoid the vanishing gradient problem, residual connections and layer normalization are applied after each sublayer (i.e., after the multi-head attention and the feedforward network).

For the multi-head attention, the residual connection is defined as:

$$\mathbf{X}' = \text{LayerNorm}(\mathbf{X} + \text{MultiHead}(\mathbf{X}, \mathbf{X}, \mathbf{X}))$$

where the input $\mathbf{X}$ is added to the output of the multi-head attention and then passed through a layer normalization function.

Similarly, for the feedforward layer:

$$\mathbf{X}'' = \text{LayerNorm}(\mathbf{X}' + \text{FFN}(\mathbf{X}'))$$

The layer normalization ensures that the output of each sublayer has zero mean and unit variance, improving model stability during training.

Key Takeaways

- **BLEU Score:** The BLEU score evaluates machine translation quality by measuring how closely a machine-generated translation matches human reference translations using n-gram precision.
- **Modified Precision:** BLEU uses modified precision to count matching n-grams, clipping the count to avoid inflating scores from repeated matches.
- **Brevity Penalty:** A brevity penalty is applied to prevent short translations from artificially inflating the BLEU score by matching fewer n-grams.
- **Final BLEU Formula:** BLEU combines brevity penalties with the geometric mean of n-gram precisions to calculate a final score for the translation quality.
- **Transformer Overview:** The Transformer model revolutionizes machine translation by eliminating recurrence in RNNs and using parallel processing through self-attention mechanisms.
- **Tokenization and Embeddings:** Text is tokenized and transformed into dense vector representations, which are then used as input to the Transformer model.
- **Positional Encoding:** Position information is added to token embeddings using sine and cosine functions to allow the model to distinguish token positions.
- **Self-Attention:** The self-attention mechanism enables each token to focus on relevant parts of the sequence, computing dependencies between tokens.
- **Multi-Head Attention:** Multiple attention heads allow the model to capture different relationships between tokens in parallel, improving its understanding of the sequence.
- **Feedforward Layers:** Each token is passed through a position-wise feedforward network to learn complex transformations independently for each token.
- **Residual Connections and Layer Normalization:** Residual connections and layer normalization stabilize training and prevent issues like vanishing gradients.