Contents

WEEK 7: INTRODUCTION TO TRANSFORMERS

From Static to Contextual Embeddings

Traditional word embeddings, such as Word2Vec and GloVec, represent each word with a single, fixed vector in a high-dimensional space. While these static embeddings have proven useful in various natural language processing tasks, they suffer from several limitations. One major drawback is their inability to capture word sense ambiguity and context-dependent meaning. For instance, the word "bank" can refer to a financial institution or the edge of a river, but static embeddings assign a single vector representation to this word, failing to distinguish between its different senses.

Polysemy, the phenomenon of a word having multiple related meanings, poses another challenge for static embeddings. Consider the word "run," which can mean to move quickly on foot, to operate or execute a program, or to seek election for a political office. Static embeddings struggle to capture these nuanced meanings, as they compress all senses of a word into a single vector. This limitation hinders the ability of models using static embeddings to accurately understand and interpret the intended meaning of words in different contexts.

To address these limitations, researchers have developed contextual embeddings, which generate dynamic vector representations of words based on their surrounding context. Models like ELMo (Embeddings from Language Models) and BERT (Bidirectional Encoder Representations from Transformers) have revolutionized the field of natural language processing by incorporating context into word embeddings. These models are trained on large corpora of text using deep neural networks, allowing them to learn complex patterns and relationships between words.

Contextual embeddings capture the semantic and syntactic information of words in a given context, enabling them to disambiguate word senses and handle polysemy effectively. For example, the contextual embedding of the word "bank" in the sentence "I deposited money at the bank" would be different from its embedding in the sentence "We sat on the bank of the river." By considering the surrounding words and the overall context, contextual embeddings provide a more accurate and nuanced representation of word meaning.

The advent of contextual embeddings has led to significant improvements in various natural language processing tasks, such as sentiment analysis, named entity recognition, and machine translation. Models like BERT have achieved state-of-the-art performance on a wide range of benchmarks, demonstrating the power of incorporating context into word representations. As research in this area continues to advance, contextual embeddings are expected to play an increasingly crucial role in enabling machines to understand and process human language more effectively.

Self-Attention Concept

Self-attention is a fundamental component of the Transformer architecture, which has revolutionized natural language processing and other domains. The self-attention mechanism allows the model to attend to different positions of the input sequence, capturing long-range dependencies and contextual information. At its core, self-attention operates on three main components: queries, keys, and values.

The query, key, and value paradigm is central to the self-attention mechanism. For each position in the input sequence, the model learns to generate a query vector, a key vector, and a value vector. The query vector represents the current position's information need, while the key vectors from all positions are compared with the query to determine the relevance of each position to the current one. The value vectors contain the actual information from each position that will be aggregated based on the attention weights.

Attention weights are computed by taking the dot product between the query vector and each key vector, followed by a softmax activation function. This normalization ensures that the attention weights sum up to 1, representing a probability distribution over the input positions. The attention weights indicate the importance of each position with respect to the current query. Positions with higher attention weights contribute more to the final representation of the current position.

The self-attention mechanism enables the model to generate context-aware representations by attending to relevant positions in the input sequence. By computing the weighted sum of the value vectors based on the attention weights, the model incorporates information from the most relevant positions, effectively capturing the context surrounding each position. This allows the model to consider the dependencies and relationships between different parts of the input, leading to more accurate and contextually rich representations.

The self-attention mechanism is highly parallelizable and computationally efficient compared to traditional recurrent neural networks. It allows for direct information flow between any two positions in the sequence, regardless of their distance. This enables the model to capture long-range dependencies effectively and handle variable-length sequences without the vanishing gradient problem. Self-attention has been successfully applied to various tasks, including machine translation, sentiment analysis, and language understanding, showcasing its versatility and effectiveness in capturing contextual information.

Transformer Architecture Basics

The Transformer architecture, introduced by Vaswani et al. in the paper "Attention Is All You Need," has revolutionized the field of natural language processing and has become the foundation for state-of-the-art models in various tasks such as machine translation, text generation, and sentiment analysis. The core components of the Transformer architecture include multi-head attention, feed-forward networks, and layer normalization, which work together to process and transform input sequences effectively.

Multi-head attention is a key component of the Transformer architecture that allows the model to attend to different parts of the input sequence simultaneously. In multi-head attention, the input is first linearly projected into multiple subspaces, each representing a different "head." Each head performs a separate attention computation, where the attention weights are calculated based on the similarity between the query, key, and value vectors. The attention weights are then used to compute a weighted sum of the value vectors, resulting in an output for each head. The outputs from all heads are concatenated and linearly transformed to obtain the final output of the multi-head attention layer. Mathematically, the attention computation for a single head can be expressed as:

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

where Q, K, and V represent the query, key, and value matrices, respectively, and d_k is the dimension of the key vectors.

Following the multi-head attention layer, the Transformer architecture employs a position-wise feed-forward network (FFN) to further process the attended features. The FFN consists of two linear transformations with a ReLU activation in between. The purpose of the FFN is to introduce non-linearity and increase the model's capacity to learn complex relationships between the attended features. The FFN is applied independently to each position in the sequence, allowing for parallel computation. The output of the FFN at position i can be expressed as:

$$\text{FFN}(x_i) = \max(0, x_i W_1 + b_1)W_2 + b_2$$

where W_1, W_2, b_1, and b_2 are learnable parameters of the FFN.

Layer normalization is applied after each sub-layer (multi-head attention and FFN) in the Transformer architecture to stabilize the training process and improve the model's convergence. Layer normalization normalizes the activations of each layer by computing the mean and variance of the activations across the features and applying a linear transformation to scale and shift the normalized values. The normalized output of a layer x can be computed as:

$$\text{LayerNorm}(x) = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \odot \gamma + \beta$$

where μ and σ^2 are the mean and variance of the activations, respectively, ϵ is a small constant for numerical stability, and γ and β are learnable scaling and shifting parameters.

Positional Encoding in Transformer Models

Positional encoding is a crucial component in transformer-based models, such as the groundbreaking Transformer architecture introduced by Vaswani et al. (2017). Unlike recurrent neural networks (RNNs) or convolutional neural networks (CNNs), transformers do not inherently capture the sequential nature of input data. To address this limitation, positional encodings are incorporated into the input embeddings, allowing the model to learn and utilize the relative positions of tokens within a sequence.

The importance of positional information lies in the fact that the meaning and context of words or tokens in a sequence often depend on their relative positions. For instance, in the sentence "The cat chased the mouse," the subject-verb-object relationship is determined by the order of the words. Without positional encodings, the transformer would treat the input as a bag of words, losing the critical sequential information. By injecting positional information into the input embeddings, the transformer can effectively capture and leverage the word order to generate more accurate and contextually relevant outputs.

One common approach to positional encoding is the use of sinusoidal functions, as proposed in the original Transformer paper. Sinusoidal encodings define a fixed set of positional encodings based on sine and cosine functions of varying frequencies. The positional encoding vector $PE_{(pos,2i)}$ for an even dimension $2i$ and position pos is given by:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Similarly, for an odd dimension $2i+1$, the positional encoding vector $PE_{(pos,2i+1)}$ is defined as:

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Here, d_{model} represents the dimensionality of the input embeddings. The sinusoidal encodings create a unique positional encoding vector for each position, allowing the model to distinguish and learn from the relative positions of tokens. The use of sine and cosine functions with varying frequencies enables the model to capture both short-term and long-term dependencies in the input sequence.

An alternative to sinusoidal encodings is the use of learned positional embeddings. Instead of using fixed sinusoidal functions, learned positional embeddings are trainable parameters that are learned during the model's training process. These embeddings are randomly initialized and updated through backpropagation, allowing the model to learn and optimize the positional representations specifically for the task at hand. Learned positional embeddings offer more flexibility and adaptability compared to fixed sinusoidal encodings, as they can capture task-specific positional patterns and dependencies. However, they also introduce additional parameters to the model, which can increase the computational complexity and memory requirements.

The choice between sinusoidal encodings and learned positional embeddings often depends on the specific task, dataset, and computational constraints. Sinusoidal encodings are computationally efficient and have been shown to perform well in various natural language processing tasks. On the other hand, learned positional embeddings may be preferred when dealing with tasks that require capturing more complex positional patterns or when the input sequences are of variable lengths. Empirical evaluations and ablation studies can help determine the most suitable positional encoding approach for a given application.

Understanding Self-Attention

Self-attention is a fundamental component of the Transformer architecture, which has revolutionized natural language processing and other domains. The key idea behind self-attention is to allow the model to attend to different positions of the input sequence to compute a representation of that sequence. This enables the model to capture long-range dependencies and contextual information effectively.

The mathematical foundation of self-attention lies in the scaled dot-product attention mechanism. Given a query matrix

$$Q$$

, a key matrix

$$K$$

, and a value matrix

$$V$$

, the attention function is computed as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where

$$d_k$$

is the dimension of the keys. The scaling factor

$$\frac{1}{\sqrt{d_k}}$$

is introduced to prevent the dot products from growing too large, which can cause the softmax function to have extremely small gradients.

One of the key advantages of self-attention is its ability to be computed in parallel. The matrix multiplications

$$QK^T$$

and the subsequent multiplication with

$$V$$

can be efficiently parallelized on modern hardware, such as GPUs. This parallel computation enables the Transformer to process input sequences much faster compared to recurrent neural networks (RNNs) or convolutional neural networks (CNNs).

Attention masks play a crucial role in self-attention, particularly in the context of sequence-to-sequence tasks like machine translation. The purpose of attention masks is to prevent the model from attending to certain positions in the input sequence. For example, in the decoder of a Transformer, future positions are masked to ensure that the model only attends to the positions up to the current time step during training. This is achieved by setting the attention scores for the masked positions to negative infinity before applying the softmax function, effectively forcing the attention weights for those positions to be zero.

Multi-head Mechanism in Transformers

The multi-head mechanism is a crucial component of the Transformer architecture, which has revolutionized natural language processing tasks. This mechanism allows the model to attend to different representation subspaces simultaneously, enabling it to capture various aspects of the input sequence. The multi-head attention is defined as:

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

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

The multi-head mechanism employs parallel attention heads, each operating on a different representation subspace. Each head performs a scaled dot-product attention, which is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

where Q, K, and V are the query, key, and value matrices, respectively, and d_k is the dimension of the key vectors. The scaling factor $\frac{1}{\sqrt{d_k}}$ is introduced to prevent the dot products from growing too large, which can lead to vanishing gradients in the softmax function.

The parallel attention heads allow the model to attend to different positions in the input sequence simultaneously, capturing diverse relationships between the elements. Each head operates on a lower-dimensional subspace, which is obtained by linearly projecting the input embeddings using the corresponding weight matrices (W_i^Q, W_i^K, and W_i^V). This enables the model to learn different representations of the input sequence, focusing on various aspects such as syntactic structure, semantic relationships, or positional dependencies.

After computing the attention outputs from each head, the information is combined by concatenating the results and applying a linear transformation using the weight matrix W^O. This combination allows the model to integrate the information from different representation subspaces, creating a rich and expressive representation of the input sequence. The multi-head mechanism enhances the model's ability to capture complex dependencies and relationships within the input, leading to improved performance on various natural language processing tasks.

The number of attention heads (h) and the dimensionality of each head (d_k and d_v) are hyperparameters that can be tuned based on the specific task and dataset. Increasing the number of heads allows the model to attend to more diverse aspects of the input, while increasing the dimensionality of each head provides more expressive power for capturing intricate relationships. However, it is essential to strike a balance between model complexity and computational efficiency, as a larger number of heads and higher dimensionality can increase the computational cost and memory requirements of the model.

Visualization and Interpretation of Transformer Attention Mechanisms

Attention mechanisms are a crucial component of transformer-based models, enabling them to effectively capture and leverage dependencies between input tokens. To gain a deeper understanding of how these models process and interpret information, researchers have developed various techniques for visualizing and analyzing attention patterns. One such approach involves examining the attention weights assigned by each attention head to different input tokens, providing insights into the model's internal workings.

Attention patterns can reveal the relationships and dependencies that the model has learned to focus on during the training process. By visualizing these patterns, researchers can identify which input tokens are most relevant to the model's predictions and how the model attends to different parts of the input sequence. For example, in a sentiment analysis task, the model may assign higher attention weights to words that strongly indicate positive or negative sentiment, while paying less attention to neutral or irrelevant tokens.

Visualizing attention patterns can also help identify potential biases or limitations in the model's learning process, such as over-reliance on certain keywords or failure to capture long-range dependencies.

Another aspect of attention visualization is the analysis of head specialization. Transformer models typically employ multiple attention heads, each of which can learn to focus on different aspects of the input. By examining the attention patterns of individual heads, researchers can uncover the specific roles and specializations that each head has developed during training. For instance, some heads may specialize in capturing syntactic relationships, such as subject-verb agreement, while others may focus on semantic relations, such as word synonymy or antonymy. Understanding head specialization can provide valuable insights into the model's ability to capture different types of linguistic information and can guide efforts to improve model performance through techniques such as head pruning or attention-based regularization.

A key distinction in attention analysis is the difference between position attention and content attention. Position attention refers to the model's ability to attend to specific positions in the input sequence, regardless of the actual content at those positions. This type of attention is particularly relevant in tasks that involve sequential or temporal dependencies, such as language modeling or time series forecasting. In contrast, content attention focuses on the semantic meaning of the input tokens, allowing the model to attend to specific words or phrases based on their relevance to the task at hand. By comparing position and content attention patterns, researchers can gain insights into how the model balances the importance of sequential order versus semantic meaning in different contexts.

Mathematically, attention weights can be represented as a matrix

$$A \in \mathbb{R}^{n \times n}$$

, where

$$n$$

is the length of the input sequence. Each element

$$a_{ij}$$

of the attention matrix represents the attention weight assigned by the model to the relationship between input tokens

$$i$$

and

$$j$$

. The attention weights are typically computed using a softmax function, ensuring that they sum to 1 for each input token:

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})}$$

where

$$e_{ij}$$

is the raw attention score computed by the model for the pair of tokens

$$(i, j)$$

. By visualizing the attention matrix

$$A$$

using techniques such as heatmaps or graph representations, researchers can gain a clear understanding of the model's attention patterns and the relationships it has learned to focus on during the training process.

Learning Objectives

- Understand the transition from embeddings to transformers
- Master the self-attention mechanism
- Grasp the core components of transformer architecture
- Visualize and interpret attention patterns