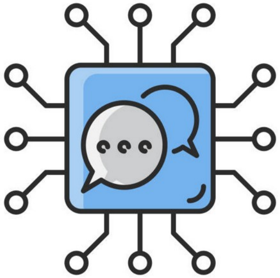
Transformers

Tushar B. Kute,
http://tusharkute.com



Seq2Seq Model Challenges

- Despite being so good at what it does, there are certain limitations of seq-2-seq models with attention:
  - Dealing with long-range dependencies is still challenging.
  - The sequential nature of the model architecture prevents parallelization. These challenges are addressed by Google Brain's Transformer concept.

Transformer

- The Transformer in NLP is a novel architecture that aims to solve sequence-to-sequence tasks while handling long-range dependencies with ease. The Transformer was proposed in the paper *Attention Is All You Need*. It is recommended reading for anyone interested in NLP.
- Quoting from the paper:
  - "The Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution."
  - Here, "transduction" means the conversion of input sequences into output sequences. The idea behind Transformer is to handle the dependencies between input and output with attention and recurrence completely.

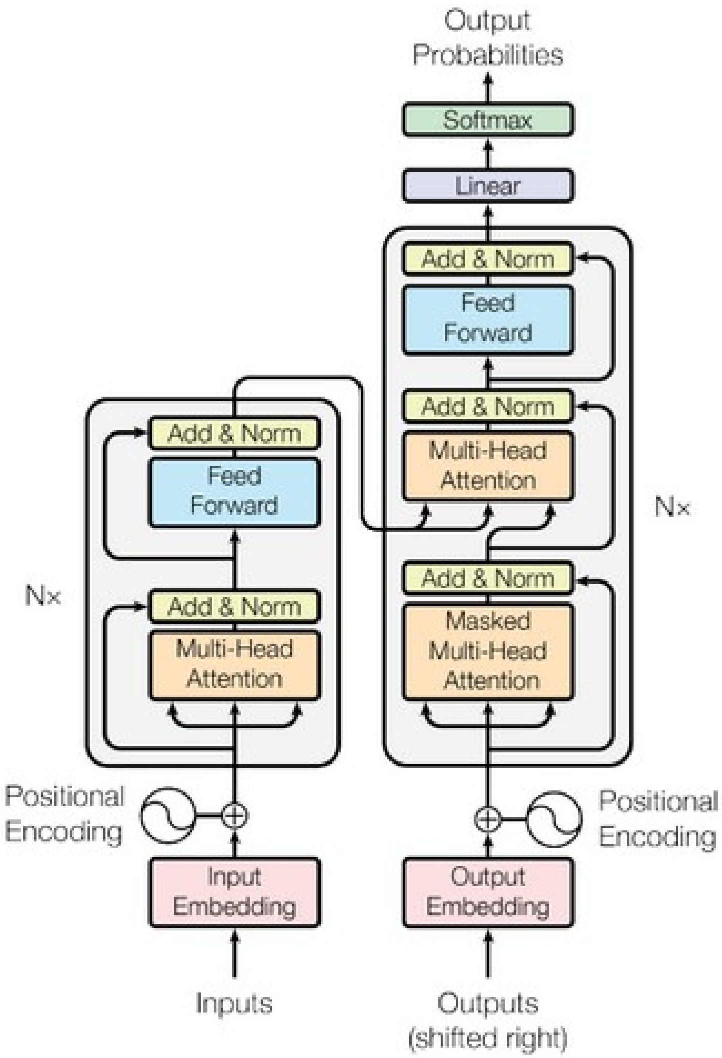
Transformer

- In the realm of neural networks, a transformer is a powerful architecture that has revolutionized the way we handle sequence-to-sequence tasks, particularly in the field of natural language processing (NLP).
- Unlike its predecessors, which relied on recurrent neural networks (RNNs) like LSTMs, transformers don't need sequential processing, making them faster and more efficient.

Transformer : Core Concepts

- Transformers ditch the sequential processing of RNNs and instead rely on a mechanism called attention.
- Attention allows the model to focus on the most relevant parts of the input sequence for each element within that sequence, considering the entire sequence at once.
- *Imagine it like attending a party where you can listen to multiple conversations simultaneously, focusing on the most interesting bits of each.*

Transformer : Model



(Source: https://arxiv.org/abs/1706.03762)

Transformer : Workflow

- Input Sentence: Start with the input text sequence (e.g., "The cat sat on the mat").
- Embeddings: Convert each word into an embedding vector.
- Positional Encoding: Add positional encodings to the word embeddings to capture word order.
- Encoder Layer:
  - Apply multi-head attention to focus on different parts of the input sequence.
  - Pass the output through a feedforward neural network.
- Decoder Layer:
  - Use masked self-attention to focus on previously generated words.
  - Apply cross-attention to focus on the relevant parts of the input sequence.
- Final Output: The decoder generates the translated or predicted text sequence.

Transformer : Step by Step

- 1. Input Embeddings:
  - The first step is converting the input sequence of words (e.g., a sentence) into numerical representations.
  - Each word is represented by an embedding vector in a continuous vector space, typically using methods like Word2Vec, GloVe, or learned embeddings.
  - Example: If the input sentence is "The cat sat on the mat", each word is converted into a fixed-size vector, resulting in a sequence of vectors.

Transformer : Step by Step

- 2. Positional Encoding:
  - Since Transformers do not have inherent sequential processing (like RNNs), positional encodings are added to the word embeddings to preserve the order of the words in the sequence.
  - The positional encoding is a vector that indicates the position of each word in the sequence. This is crucial because the model should know whether a word appears at the beginning, middle, or end of the sentence.
- Mathematically, the positional encoding is often calculated using sine and cosine functions, ensuring unique encodings for different positions.

Transformer : Step by Step

- 3. Encoder (Self-Attention Mechanism):

  The Encoder part of the Transformer consists of a stack of layers. Each layer has two main components:

  – Self-Attention: This mechanism allows the model to focus on different parts of the input sequence when encoding a word. It computes a weighted sum of all input words (including the word itself) based on their relationships.

  – Self-attention computes three vectors for each word:

    - Query (Q): A vector representing the current word we want to focus on.
    - Key (K): A vector representing every other word in the sequence that the current word should pay attention to.
    - Value (V): A vector representing the information to be passed along.

Transformer : Step by Step

  - The attention score is calculated by taking the dot product of the Query vector and the Key vector, followed by a softmax operation to normalize the scores. The higher the score, the more attention is given to that word.
- Multi-Head Attention:
  - Instead of using a single attention mechanism, multiple attention heads are used in parallel. This allows the model to focus on different parts of the sentence simultaneously, capturing various aspects of word relationships.
  - Each attention head performs the attention operation, and the results are concatenated and passed through a linear layer.

Transformer : Step by Step

- Feedforward Neural Network:
  - After the attention mechanism, the output is passed through a fully connected feedforward neural network (with activation functions like ReLU) for further processing.
- Each encoder layer consists of:
  - Multi-head self-attention
  - Feedforward neural network
  - Residual connections and Layer Normalization to help with training stability.

Transformer : Step by Step

- 4. Decoder (Self-Attention + Cross-Attention):
  - The Decoder generates the output sequence (for example, in machine translation, the translated sentence). The decoder also has multiple layers, each consisting of:
    - Masked Self-Attention:

      Similar to self-attention in the encoder, but in the decoder, it is "masked" to ensure that each word only attends to previously generated words, preserving the autoregressive nature of sequence generation.

Transformer : Step by Step

- Cross-Attention (Encoder-Decoder Attention):
  - The decoder receives attention not only from its previous layer's output but also from the encoder's output. This enables the decoder to focus on relevant parts of the input sequence while generating the output sequence.
  - The attention scores are computed between the Query vector (from the decoder) and the Key vector (from the encoder) to decide how much attention to give to the encoder's output.
- Feedforward Neural Network:
  - Similar to the encoder, after the attention operation, the output is passed through a feedforward neural network.

Transformer : Step by Step

- 5. Output Layer:
  - The decoder output is passed through a final linear layer and softmax activation to predict the next word in the sequence (for generation tasks like text translation or text completion).
  - The predicted word can be fed back into the decoder in the next step to predict subsequent words (autoregressive process).

Transformer : Step by Step

- 6. Final Output:
  - The Transformer model’s final output is a sequence of words (in NLP tasks, this could be a translation, a summary, or the next word prediction).

Transformer : Step by Step

- This step-by-step process allows the Transformer to effectively model dependencies in sequences without the need for recurrent connections, making it highly parallelizable and efficient for both training and inference.

Transformer : Benefits

- Self-Attention:
  - Allows the model to focus on different parts of the input sequence simultaneously.
- Parallelization:
  - Unlike RNNs, which process tokens sequentially, transformers process the entire sequence at once, making them more efficient.
- Scalability:
  - Transformers can handle longer sequences and large datasets effectively.
- Multi-Head Attention:
  - Enables the model to focus on different aspects of the sequence in parallel.

Transformer: Limitations

- Transformer is undoubtedly a huge improvement over the RNN based seq2seq models. But it comes with its own share of limitations:
  - Attention can only deal with fixed-length text strings. The text has to be split into a certain number of segments or chunks before being fed into the system as input
  - This chunking of text causes context fragmentation. For example, if a sentence is split from the middle, then a significant amount of context is lost.
  - In other words, the text is split without respecting the sentence or any other semantic boundary.

TransformerXL

- Transformer XL, meaning "extra long," is an innovative neural network architecture based on the Transformer architecture specifically designed to overcome the limitations of standard Transformers in capturing long-range dependencies in sequences.

TransformerXL

- Transformer architectures can learn longer-term dependency. However, they can't stretch beyond a certain level due to the use of fixed-length context (input text segments).
- A new architecture was proposed to overcome this shortcoming in the paper – *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context.*
- In this architecture, the hidden states obtained in previous segments are reused as a source of information for the current segment.
- It enables modeling longer-term dependency as the information can flow from one segment to the next.

Transformer: Challenges

- Fixed-length context: Standard Transformers can only consider a limited context window while processing sequences, hindering their ability to capture relationships between distant words. Imagine trying to understand the meaning of a sentence without considering the context of the entire paragraph.
- Temporal coherence: Processing long sequences often leads to context fragmentation, where the relationships between distant words are lost, affecting the coherence of the overall representation. It's like having a scrambled puzzle where the pieces don't quite fit together.

TransformerXL: Solutions

- Segment-level recurrence:
  - Transformer XL introduces a segment-level recurrence mechanism that maintains information across consecutive segments.
  - This allows the model to "remember" and utilize context from previous segments while processing the current one, effectively extending the effective context window.
  - Think of it as having a running memory that keeps track of the story so far.

TransformerXL: Solutions

- Relative positional encoding:
  - Instead of absolute positional encoding used in standard Transformers, Transformer XL employs relative positional encoding.
  - This encodes the relationships between words based on their relative positions, reducing the dependence on absolute word order and improving the model's ability to handle long sequences.
  - Imagine using relative directions like "two steps ahead" or "three places before" instead of absolute numbers to navigate a map.

TransformerXL: Step by Step

- 1. Input Representation
  - Transformer-XL, like the original Transformer, starts by converting input tokens into dense vectors (embeddings). Additionally, it uses relative positional encodings to better handle longer sequences.
- 2. Segment-Level Recurrence Mechanism
  - The key innovation in Transformer-XL is the segment-level recurrence mechanism. Instead of processing the entire sequence at once, Transformer-XL processes it in segments and carries over information from previous segments to handle long-term dependencies.

TransformerXL: Step by Step

- 2. Segment-Level Recurrence Mechanism
  - Segmenting Input:
    - The input sequence is divided into fixed-length segments. For example, a sequence of length 1000 might be divided into 10 segments of 100 tokens each.
  - Processing Segments:
    - Each segment is processed one at a time. The output of one segment is used to help process the next segment.
  - Recurrent Mechanism:
    - For segment ii, the hidden states from segment i–1i–1 are carried over and used as a memory. This allows the model to have a longer effective context length without the need to increase the computational complexity.

TransformerXL: Step by Step

- 3. Relative Positional Encoding
  - Traditional Transformers use absolute positional encodings, which limit their ability to generalize to longer sequences.
  - Transformer-XL uses relative positional encodings, which encode the relative distance between positions rather than their absolute positions.
  - This helps in modeling longer-term dependencies effectively.

TransformerXL: Step by Step

- 3. Relative Positional Encoding
  - Embedding Tokens and Positions:
    - Token embeddings are created for each token in the segment.
    - Relative positional encodings are calculated based on the distance between tokens within the segment.
  - Calculating Attention Scores:
    - Attention scores are computed using both token embeddings and relative positional encodings. This helps the model to understand the relative positions of tokens more effectively.

TransformerXL: Step by Step

- 4. Multi-Head Self-Attention with Recurrence
  - Each layer of the Transformer-XL model consists of multi-head self-attention mechanisms and feed-forward networks, similar to the original Transformer, but with the addition of recurrent connections.

TransformerXL: Step by Step

- 4. Multi-Head Self-Attention with Recurrence
  - Self-Attention Calculation:
    - For each segment, the self-attention mechanism calculates the attention scores using the token embeddings and relative positional encodings.
    - The attention scores are then used to compute a weighted sum of the value vectors, resulting in the self-attention output.

TransformerXL: Step by Step

- 4. Multi-Head Self-Attention with Recurrence
  - Incorporating Memory:
    - The hidden states from the previous segment (memory) are included in the self-attention calculation for the current segment.
    - This helps in maintaining a longer context and capturing dependencies beyond the current segment.
  - Feed-Forward Neural Network:
    - The self-attention output is passed through a feed-forward neural network, which consists of two linear transformations with a ReLU activation in between.

TransformerXL: Step by Step

- 4. Multi-Head Self-Attention with Recurrence
  - Layer Normalization and Residual Connections:
    - Residual connections and layer normalization are applied to the outputs of the self-attention mechanism and the feed-forward network to improve stability and convergence.

TransformerXL: Step by Step

- 5. Training and Inference
- The training and inference procedures in Transformer-XL are similar to those in the original Transformer, with additional steps to handle the memory mechanism.
  - Training:
    - Input Preparation:
      - The input sequence is divided into segments, and the memory states from previous segments are initialized (usually to zeros for the first segment).
    - Forward Pass:
      - Each segment is processed sequentially, with memory states being carried over to the next segment.

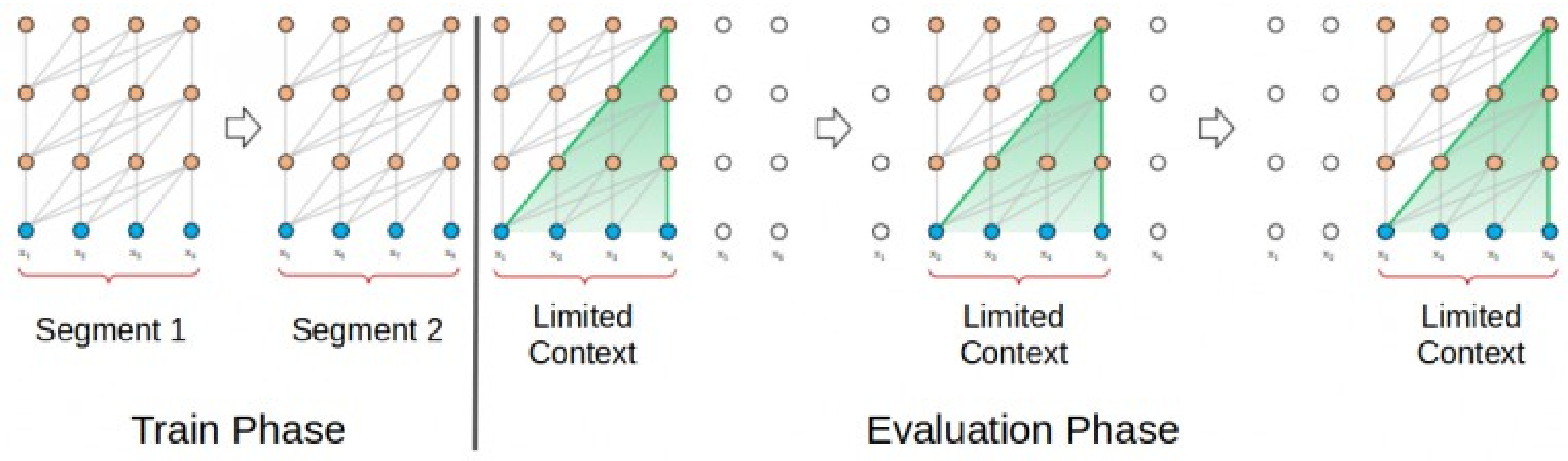
TransformerXL: Step by Step

- Training:
  - Loss Calculation:
    - The output logits are used to calculate the loss (e.g., cross-entropy loss for language modeling tasks).
  - Backpropagation:
    - Gradients are calculated and weights are updated using backpropagation through time (BPTT), which also considers the recurrent connections.

TransformerXL: Step by Step

- Inference:
  - Input Preparation:
    - Similar to training, the input sequence is divided into segments, and memory states are initialized.
  - Generating Output:
    - Each segment is processed sequentially, generating tokens one by one. The memory states are carried over to maintain context.
  - Recurrent Memory Update:
    - Memory states are updated at each step to ensure that the model can generate long sequences effectively.

Using Transformer for Language Modeling



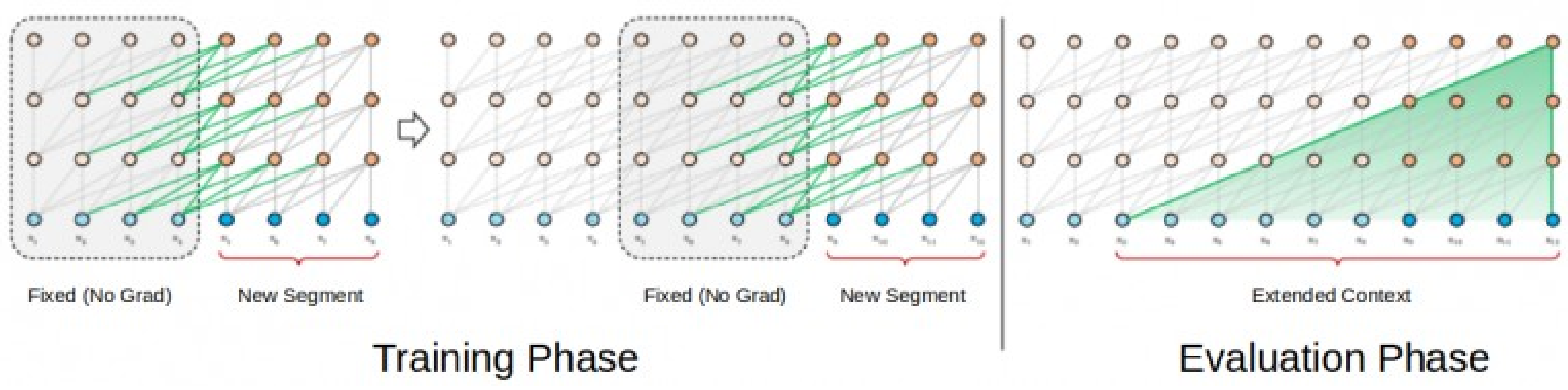
Transformer Model with a segment length of 4 (Source: https://arxiv.org/abs/1901.02860)

Using Transformer for Language Modeling

- This architecture doesn’t suffer from the problem of vanishing gradients. But the context fragmentation limits its longer-term dependency learning.
- During the evaluation phase, the segment is shifted to the right by only one position. The new segment has to be processed entirely from scratch.
- This evaluation method is unfortunately quite compute-intensive.

Using Transformer for Language Modeling

- During the training phase in Transformer-XL, the hidden state computed for the previous state is used as an additional context for the current segment.
- This recurrence mechanism of Transformer-XL takes care of the limitations of using a fixed-length context.



Transformer XL Model with a segment length of 4

TransformerXL: Summary

- Transformer-XL extends the original Transformer model by introducing segment-level recurrence and relative positional encodings, enabling it to handle longer sequences and capture long-term dependencies more effectively.
- The key steps include segmenting the input, using memory from previous segments, calculating self-attention with relative positional encodings, and processing segments sequentially during training and inference.

Thank you

@mitu_skillologies
@mITuSkillologies
@mitu_group
@mitu-skillologies
@MITUSkillologies

kaggle
@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in
tushar@tusharkute.com