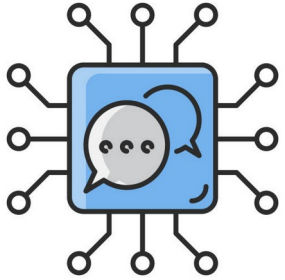
Transformers

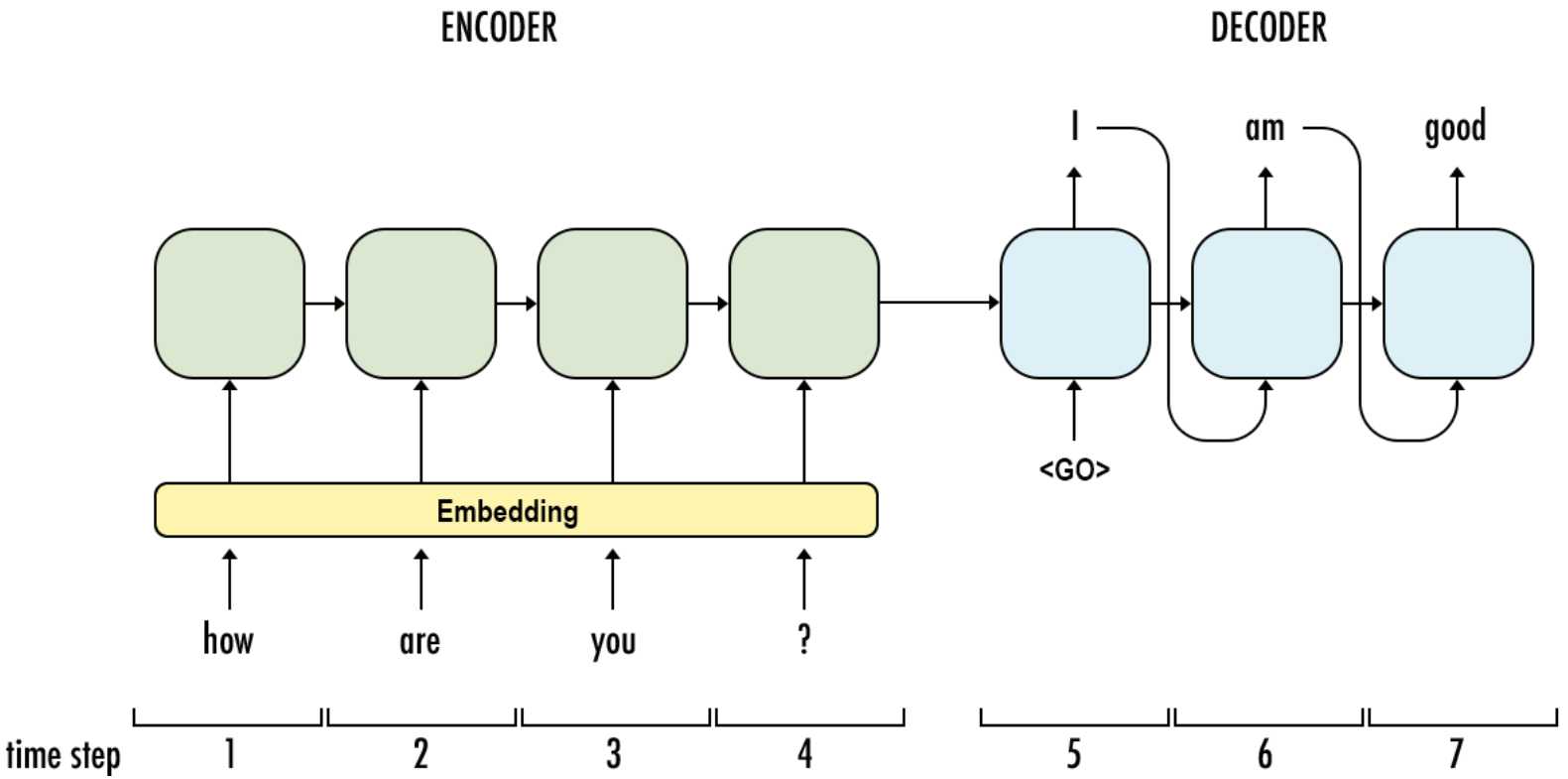
Tushar B. Kute,
http://tusharkute.com



Sequence to Sequence Models

- Sequence-to-sequence (seq2seq) models in NLP are used to convert sequences of Type A to sequences of Type B.
- For example, translation of English sentences to German sentences is a sequence-to-sequence task.
- Recurrent Neural Network (RNN) based sequence-to-sequence models have garnered a lot of traction ever since they were introduced in 2014.
- Most of the data in the current world are in the form of sequences – it can be a number sequence, text sequence, a video frame sequence or an audio sequence.

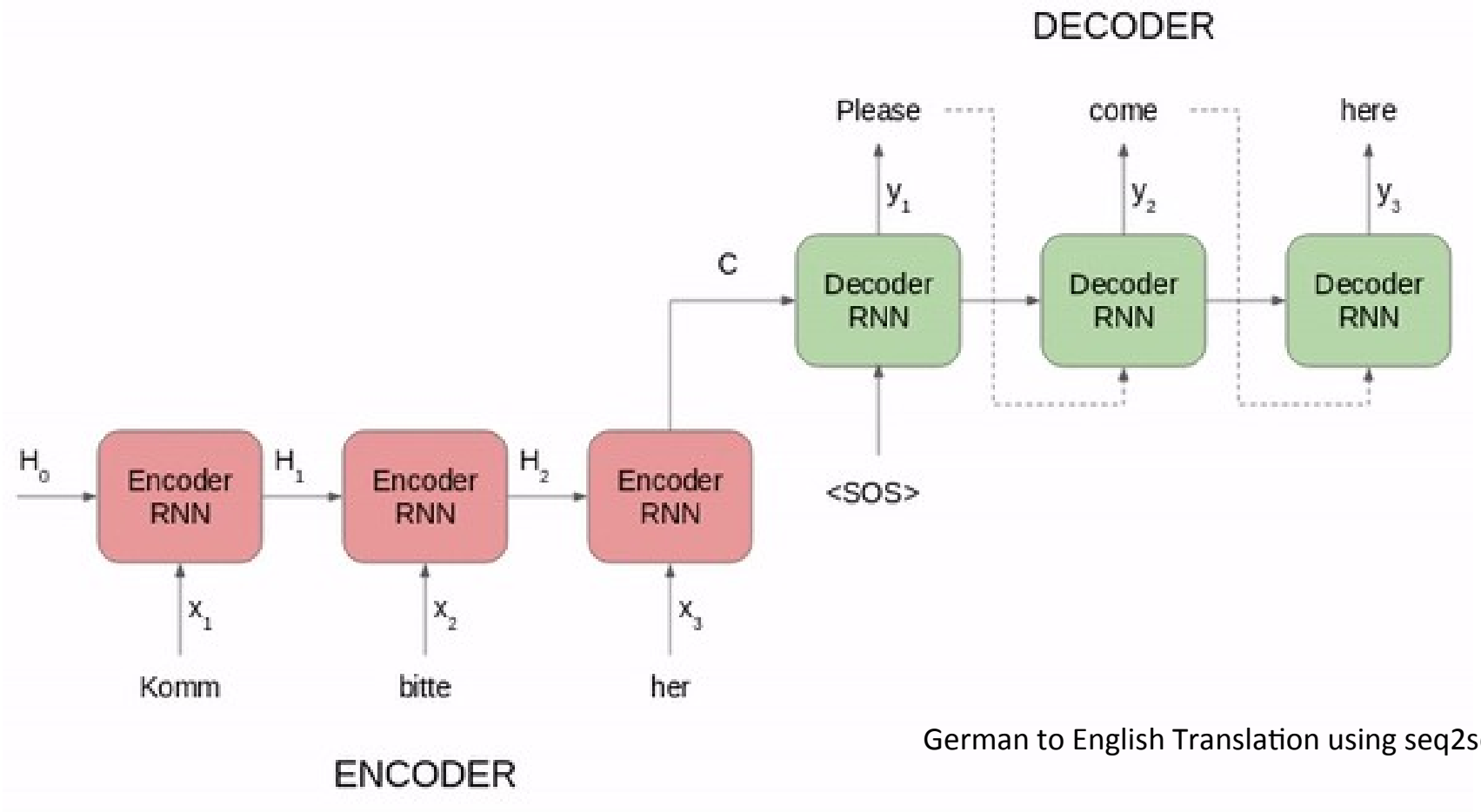
Sequence to Sequence Models



Sequence to Sequence Models

- The performance of these seq2seq models was further enhanced with the addition of the Attention Mechanism in 2015. How quickly advancements in NLP have been happening in the last 5 years – incredible!
- These sequence-to-sequence models are pretty versatile and they are used in a variety of NLP tasks, such as:
  - Machine Translation
  - Text Generation
  - Text Summarization
  - Speech Recognition
  - Question-Answering System, and so on

Sequence to Sequence Models



German to English Translation using seq2seq

Sequence to Sequence Models

- The above seq2seq model is converting a German phrase to its English counterpart.
- Let's break it down:
  - Both Encoder and Decoder are RNNs
  - At every time step in the Encoder, the RNN takes a word vector (xi) from the input sequence and a hidden state (Hi) from the previous time step
  - The hidden state is updated at each time step

Sequence to Sequence Models

- The hidden state from the last unit is known as the context vector. This contains information about the input sequence.
- This context vector is then passed to the decoder and it is then used to generate the target sequence (English phrase)
- If we use the Attention mechanism, then the weighted sum of the hidden states are passed as the context vector to the decoder.

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.

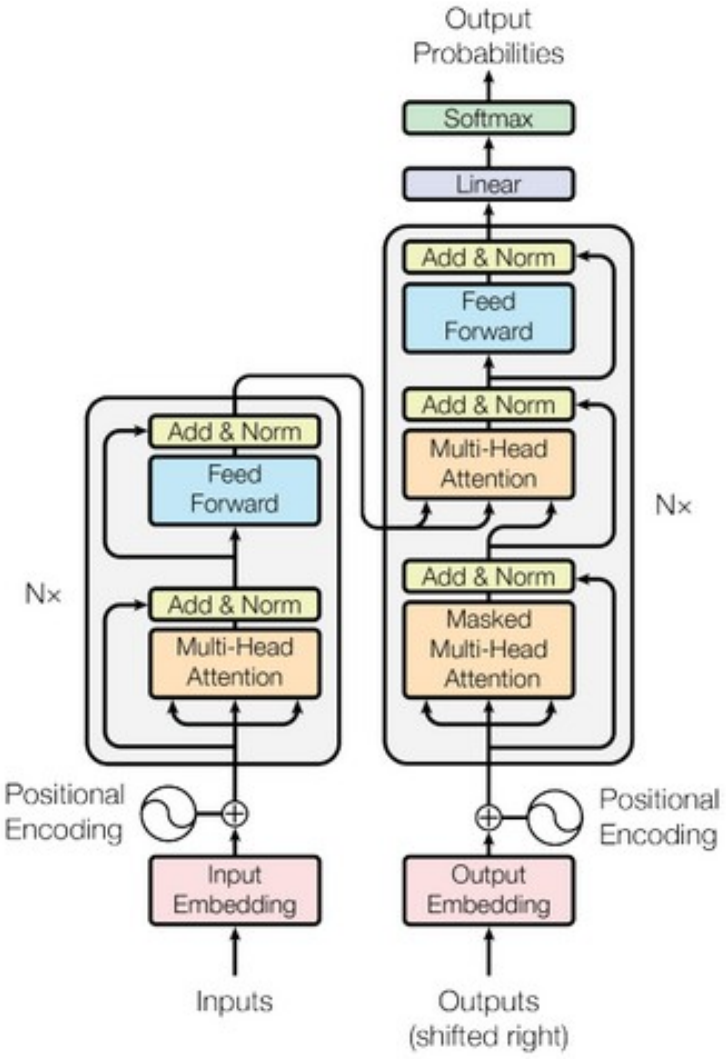
Transformer

- 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 : Step by Step

- 1. Input Representation
  - The first step is to convert the input tokens (words, subwords, or characters) into dense vectors, known as embeddings.
  - Additionally, positional encodings are added to these embeddings to incorporate the order of tokens since the model does not inherently capture sequence order.

Transformer : Step by Step

- 2. Encoder-Decoder Architecture
  - The Transformer consists of an encoder and a decoder, each made up of several layers.
  - Typically, there are six layers in both the encoder and decoder, but this number can be varied.
  - Encoder
    - Each encoder layer has two main components:
      - Multi-Head Self-Attention Mechanism
      - Feed-Forward Neural Network

Transformer : Step by Step

- Multi-Head Self-Attention:
  - The self-attention mechanism allows the model to focus on different parts of the input sentence when encoding a particular word.
  - The input embeddings are projected into three vectors: Query (Q), Key (K), and Value (V).
  - The attention scores are calculated by taking the dot product of the Query with all Keys, dividing by the square root of the dimension of the Key (for scaling), and applying a softmax function to obtain weights.

Transformer : Step by Step

- Multi-Head Self-Attention:
  - These weights are used to sum the corresponding Values, producing an output vector for each word.
  - Multi-head attention means this process is done multiple times in parallel with different learned projections.

Transformer : Step by Step

- Feed-Forward Neural Network:
  - A fully connected feed-forward network is applied to each position independently and identically.
  - This consists of two linear transformations with a ReLU activation in between.
- Each encoder layer also includes residual connections around each sub-layer, followed by layer normalization.

Transformer : Step by Step

- Decoder
  - The decoder is similar to the encoder but has an additional layer to handle the encoder's output and its own self-attention mechanism.
  - Each decoder layer has three main components:
    - Masked Multi-Head Self-Attention Mechanism
    - Multi-Head Attention over Encoder Output
    - Feed-Forward Neural Network

Transformer : Step by Step

- Masked Multi-Head Self-Attention:
  - Similar to the encoder's self-attention, but with a mask to prevent attending to future positions, ensuring the prediction for position ii can depend only on known outputs at positions less than ii.
- Multi-Head Attention over Encoder Output:
  - This attention mechanism takes the encoder’s output as Key and Value and the decoder's own output as Query.
  - This allows the decoder to focus on relevant parts of the input sequence when generating the output.

Transformer : Step by Step

- Feed-Forward Neural Network:
  - Similar to the encoder, a fully connected feed-forward network is applied to each position independently.
- Like the encoder, the decoder layers also include residual connections and layer normalization.

Transformer : Step by Step

- 3. Final Linear and Softmax Layer
  - The final layer of the decoder is followed by a linear transformation and a softmax function to convert the decoder's output to probabilities over the target vocabulary, generating the next token in the sequence.

Transformer : Step by Step

- 1. Input Tokenization and Embedding:
  - Tokenize the input text into subwords/words.
  - Convert tokens to embeddings and add positional encodings.
- 2. Pass through Encoder:
  - Each token embedding is passed through the stack of encoder layers.
  - Within each encoder layer, compute self-attention scores and apply them to obtain a weighted sum of values.
  - Apply the feed-forward network to the self-attention outputs.
  - Use residual connections and layer normalization at each sub-layer.

Transformer : Step by Step

- 3. Prepare Decoder Input:
  - For training, the target sequence is offset by one position (known as teacher forcing).
  - For inference, start with a special start token and generate tokens step-by-step.
- 4. Pass through Decoder:
  - Each decoder layer processes the embedded target tokens using masked self-attention, cross-attention with the encoder output, and a feed-forward network.
  - Use residual connections and layer normalization at each sub-layer.

Transformer : Step by Step

- 5. Generate Output:
  - The final decoder layer outputs are transformed to logits using a linear layer.
  - Apply softmax to obtain probability distributions over the target vocabulary.
  - Select the most probable token as the next output (during inference) or compare against the actual next token (during training).
- 6. Repeat for Entire Sequence:
  - During training, the entire target sequence is processed at once.
  - During inference, repeat steps 3-5 until an end token is generated or the maximum sequence length is reached.

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

- Parallelization:
  - Unlike RNNs, transformers can be parallelized, meaning they can be processed on multiple GPUs or CPUs at the same time, significantly speeding up training and inference.
- Long-range dependencies:
  - Transformers can effectively capture relationships between distant elements in a sequence, something RNNs struggle with due to their sequential nature.
- State-of-the-art performance:
  - Transformers have achieved remarkable results in various NLP tasks, including machine translation, text summarization, and question answering, making them the go-to architecture for many applications.

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.

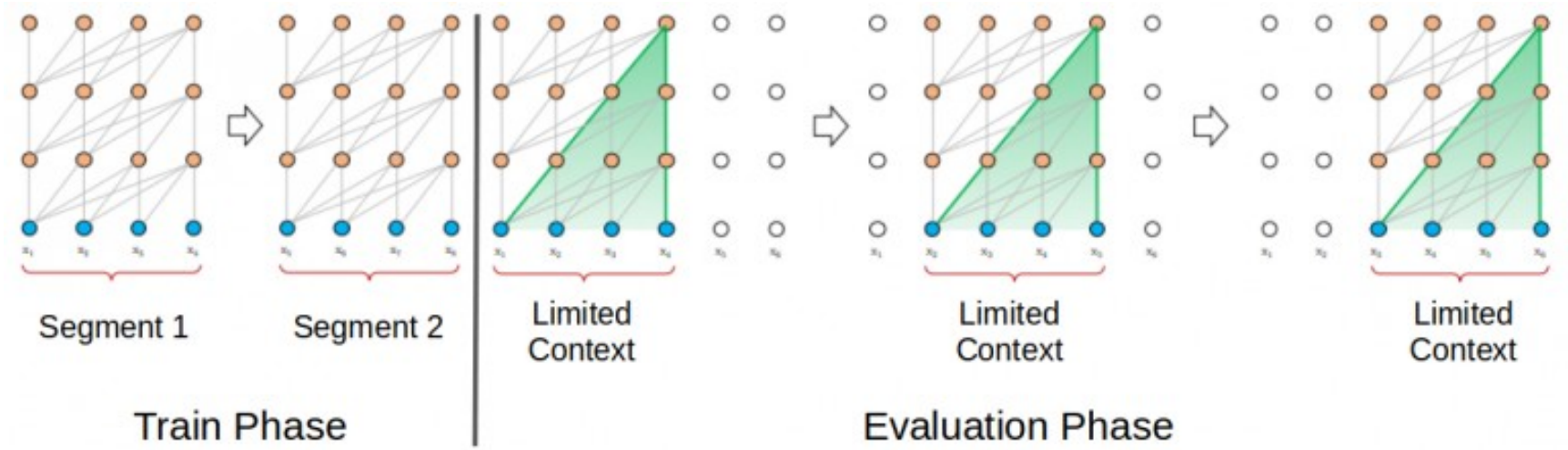
TransformerXL: Step by Step

- 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



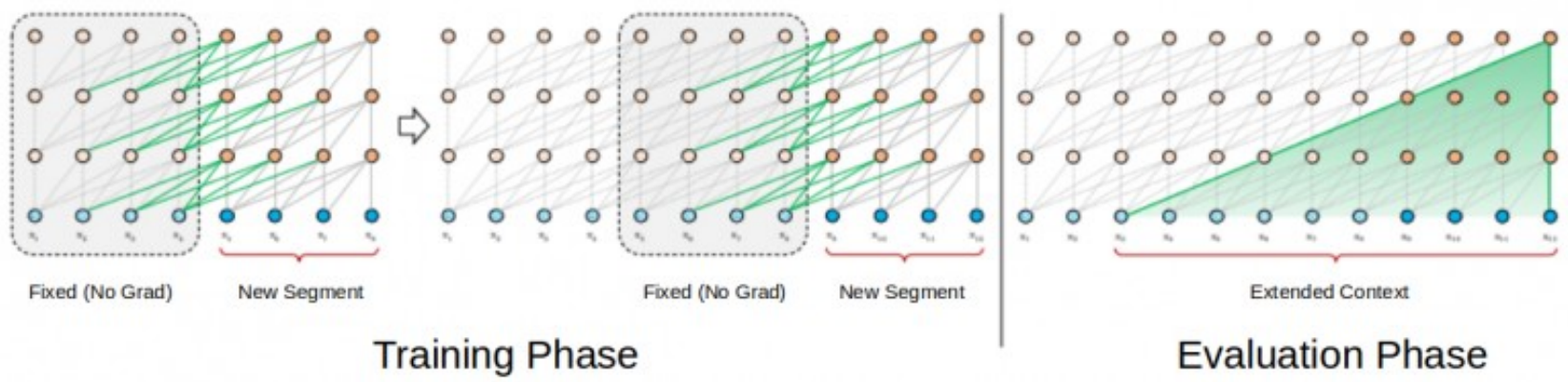
Transformer Model with a segment length of 4 (Source: https://arxiv.org/abs/1901.02860)

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

This presentation is created using LibreOffice Impress 7.4.1.2, can be used freely as per GNU General Public License

@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