STEP 0: TOWARDS GENAI

Sachin Shivakalimath
sachin.smath@gmail.com
9449764697

Seq2Seq Modeling

1. One-to-One

- **Definition:** Each input corresponds to a single output.
- **Example:** Traditional classification tasks like image classification.
  - **Input:** A single image.
  - **Output:** A single label (e.g., "cat" or "dog").

2. One-to-Many

- **Definition:** A single input generates multiple outputs.
- **Example:** Text generation or image captioning.
  - **Input:** A single image.
  - **Output:** A sequence of words describing the image (e.g., "A cat sitting on the mat").

Seq2Seq Modeling

3. Many-to-One

- **Definition:** Multiple inputs result in a single output.
- **Example:** Sentiment analysis or sequence classification.
  - **Input:** A sequence of words (e.g., "This movie is fantastic").
  - **Output:** A single label (e.g., "Positive sentiment").

4. **Many-to-Many**

- **Definition:** A sequence of inputs maps to a sequence of outputs. Can be synchronous (input and output have the same length) or asynchronous (different lengths).
- **Example 1:** Machine Translation (synchronous).**Input:** A sequence of words in one language (e.g., "How are you?").
- **Output:** A sequence of words in another language (e.g., "Comment ça va?").
- **Example 2:** Video frame labeling (asynchronous).**Input:** Frames of a video.
- **Output:** Labels for each frame.

Encoder Decoder Architecture

- The **Encoder-Decoder architecture** is a neural network design commonly used in **sequence-to-sequence (Seq2Seq) tasks**, where the input and output are sequences of variable lengths. This architecture is a key component in tasks like machine translation, text summarization, and speech-to-text systems.

Key Components of Encoder Decoder

- **Encoder:**
- The **encoder** processes the input sequence and compresses its information into a fixed-size context vector (also called the **latent representation** or **hidden state**).
- It typically uses **Recurrent Neural Networks (RNNs)** like LSTMs or GRUs, or **Transformers**.
- The encoder's job is to summarize the input sequence into a meaningful representation that the decoder can understand.

Steps in the Encoder:

- Input sequence → RNN/Transformer layers → **Context vector**

- **Decoder:**
- The **decoder** takes the context vector from the encoder and generates the output sequence, one step at a time.
- It uses its own hidden states and previously generated tokens to predict the next token.
- The decoder is also usually implemented using RNNs, LSTMs, GRUs, or Transformers.
- **Steps in the Decoder:**
- Context vector + previous outputs → RNN/Transformer layers → Output sequence

How it works

- Let’s consider an example of **English-to-French translation** (e.g., "I am happy" → "Je suis heureux").
- **Encoder:**
  - The input sentence "I am happy" is tokenized and passed through the encoder.
  - The encoder processes the input step-by-step and generates a context vector summarizing the sentence.
- **Decoder:**
  - The decoder takes the context vector as input and generates the output sequence, one word at a time.
  - For example:
    - At time step 1: Generate "Je."
    - At time step 2: Use "Je" and the context vector to generate "suis."
    - At time step 3: Use "Je suis" and the context vector to generate "heureux."

Application of Encoder Decoder

- The Encoder-Decoder architecture is used in:
  - **Machine Translation:** Translating text from one language to another.
  - **Text Summarization:** Generating concise summaries from long texts.
  - **Speech-to-Text:** Converting speech audio into written text.
  - **Image Captioning:** Describing the content of an image in natural language.
  - **Dialogue Systems:** Generating responses in conversational agents or chatbots.

Strengths and Limitations of Encoder Decoder

Strengths :-

- Handles variable-length input and output sequences.
- Encapsulates the input sequence into a meaningful latent representation.
- Works well with complex sequence-related tasks when combined with attention mechanisms.

Limitations:

- **Fixed Context Vector (Without Attention):**
  - Compressing all input sequence information into a fixed-size vector can lead to loss of information, especially for long sequences.
- **Sequential Nature:**
  - The decoder generates tokens step-by-step, making the process slower than parallelizable architectures (e.g., Transformers).
- **Training Challenges:**
  - Training encoder-decoder models can be computationally expensive, especially for long sequences.

Encoder-Decoder with Attention:

- A significant improvement to this architecture was the introduction of **attention mechanisms**, where the decoder dynamically focuses on relevant parts of the input sequence. This overcomes the limitation of relying on a single context vector.
- For example, in **machine translation**, attention allows the model to focus on specific words in the input sentence while generating each word in the output.

Why Use Attention?

- **Problem with Fixed-Size Context Vector:**
  - In the vanilla Encoder-Decoder architecture, the encoder encodes the entire input sequence into a single context vector.
  - For long sequences, this can lead to loss of important details because the fixed-size context vector cannot store all the necessary information.
- **Solution:**
  - Attention enables the decoder to look at **different parts of the input sequence** (produced by the encoder) at each decoding step.
  - Instead of relying solely on a fixed-size vector, the decoder assigns **weights** to the encoder's hidden states, indicating their relevance to the current decoding step.

How Attention Works:

Let's break it down step-by-step using an **English-to-French translation** example:

1. Encoder:

- The input sequence (e.g., "I am happy") is tokenized and processed by the encoder.
- The encoder outputs:
  - **Hidden states:** One for each token in the input sequence. These represent the contextualized information for each token.
  - Example: For "I am happy", the encoder might produce hidden states: h_1,h_2,h_3.

Attention Mechanism in the Decoder:

- At each decoding step, the decoder:
- **Calculates Attention Weights:**
  - For each input token, a score is computed to determine how relevant it is to the current decoding step.
  - These scores are typically computed using a function of the encoder hidden states (h_1,h_2,h_3) and the decoder's current state (s_t):

$$\textbf{score}(h_i, s_t) = f(h_i, s_t)$$

  Common scoring functions:

  - Dot product: $\text{score}(h_i, s_t)=h_i \cdot s_t$
  - Additive (Bahdanau attention): A learned neural network combines h_i and s_t
  - Scaled dot product (used in Transformers): Scales the dot product for numerical stability.

Contd…

How Attention Works:

2. **Softmax Normalization:**
   - The scores are passed through a softmax function to generate attention weights ($\alpha_1, \alpha_2, \alpha_3$):

$$\alpha_i = \frac{\exp(score(h_i, s_t))}{\sum_j \exp(score(h_j, s_t))}$$

   These weights sum to 1 and represent the importance of each input token.

3. **Weighted Sum of Encoder States:**
   - The encoder hidden states are combined using the attention weights to create a context vector for the current decoding step:

$$context_t = \sum_i \alpha_i h_i$$

Contd...

How Attention Works:

4. Decoder:

- The decoder uses the context vector ($context_t$) and its own hidden state (s_t) to predict the next word in the output sequence.
- For example:
  - At step 1, the decoder predicts "Je" using context1context_1context1.
  - At step 2, the decoder predicts "suis" using context2context_2context2.

This process continues until the entire output sequence is generated.

Visualization of Attention

Imagine translating "I am happy" to "Je suis heureux". At each decoding step:

1. While generating "Je", the decoder assigns higher weights to "I".

2. While generating "suis", the decoder focuses on "am".

3. While generating "heureux", the decoder focuses on "happy".

The attention mechanism helps the model dynamically focus on the most relevant words, improving the quality of translations, especially for longer sentences.

Attention Score Functions

- Some commonly used Score functions

1. **Dot Product:**

$$score(h_i, s_t) = h_i \cdot s_t$$

   - Simple and efficient.
   - Used in early attention models.

2. **Additive (Bahdanau Attention):**

$$score(h_i, s_t) = V^T \tanh(W[h_i; s_t])$$

   - Combines the encoder's hidden states and the decoder's state using a learned weight matrix W.
   - Introduced in Bahdanau et al.'s 2015 paper ("Neural Machine Translation by Jointly Learning to Align and Translate").

Contd…

Attention Score Functions

3. **Scaled Dot Product (used in Transformers):**

$$score(h_i, s_t) = \frac{h_i \cdot s_t}{\sqrt{d_k}}$$

- Scales the dot product by the square root of the hidden size (d_k) for numerical stability.
- Used in the Transformer model.

Benefits of Attention

- **Handles Long Sequences Better:**
  - Avoids the limitations of a fixed-size context vector by dynamically attending to relevant parts of the input.
- **Improves Interpretability:**
  - Attention weights can be visualized to understand which input tokens the model focused on while generating each output token.
- **Boosts Performance:**
  - Especially effective for tasks like machine translation, text summarization, and speech recognition.

Applications of Encoder-Decoder with Attention

- **Machine Translation:**
  - Translate sentences from one language to another.
- **Text Summarization:**
  - Generate a concise summary of a long article.
- **Image Captioning:**
  - Use an image encoder (like a CNN) to extract features and a decoder with attention to generate captions.
- **Speech Recognition:**
  - Convert audio sequences into text.
- **Dialogue Systems:**
  - Generate contextually relevant responses in chatbots.

Transformers: The Core Idea

- Transformers are a type of neural network architecture introduced in the groundbreaking paper **"Attention Is All You Need" (Vaswani et al., 2017)**. They have revolutionized Natural Language Processing (NLP) and other sequence-related tasks by replacing traditional recurrent or convolutional neural networks with a mechanism called **Self-Attention**.

Key Features of Transformers

- **Parallel Processing:**
  - Unlike RNNs or LSTMs, which process tokens sequentially, Transformers process all tokens simultaneously, enabling faster training and inference.
- **Self-Attention Mechanism:**
  - The core of Transformers, **self-attention** allows the model to weigh the importance of different words (or tokens) in a sequence relative to each other.
- **Scalability:**
  - Transformers scale efficiently to large datasets and sequences, making them ideal for training massive models like BERT, GPT, and T5.

Components of Transformers

1. Input Embedding

- **Embedding Layer:**
  - Each input token is converted into a dense vector (word embedding).
- **Positional Encoding:**
  - Since Transformers process all tokens simultaneously, they don't have a natural sense of order. Positional encoding is added to embeddings to encode sequence information.

2. Multi-Head Self-Attention

- **What It Does:**
  - Each token attends to all other tokens in the sequence to compute contextualized representations.
- **Steps:**
  - Compute three vectors: **Query (Q)**, **Key (K)**, and **Value (V)** for each token.
  - Calculate attention scores between tokens using dot product of Q and K, scaled by the dimension d_k, and apply softmax:

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

  - Perform this process in multiple heads (subspaces), allowing the model to focus on different aspects of the sequence simultaneously.

Contd…

Components of Transformers

3. Feed-Forward Neural Network

- After self-attention, each token’s representation is passed through a feed-forward neural network (applied independently to each token).

4. Add & Normalize

- Residual connections and layer normalization are applied to stabilize training and improve gradient flow.

5. Stacking Layers

- The Transformer consists of **stacked layers** (e.g., 6, 12, or more) of attention and feed-forward networks, enabling deep learning of complex relationships.

6. Output Layer

- For tasks like text generation, classification, or translation, a task-specific head is added to the output.

Architecture Overview

Transformers consist of two main parts:

- **A. Encoder**
- Encodes the input sequence into a contextualized representation.
- Used for tasks like sentence classification or BERT (Bidirectional Encoder Representations from Transformers).

- **B. Decoder**
- Decodes the encoder’s output into a target sequence.
- Used for tasks like machine translation or GPT (Generative Pre-trained Transformer).
- For sequence-to-sequence tasks like translation, both encoder and decoder are used.

Key Innovations in Transformers

- **Self-Attention:**
  - Models relationships between tokens regardless of their distance in the sequence.
- **Positional Encoding:**
  - Adds information about token positions in the sequence.
- **Multi-Head Attention:**
  - Enables the model to attend to different parts of the sequence in parallel.
- **Parallelization:**
  - Removes sequential dependencies, making training faster and more efficient on GPUs.

Advantages of Transformers

- **Scalability:** Efficiently scales to large datasets and sequences.
- **Parallelization:** Faster training compared to RNNs/LSTMs.
- **Global Context:** Self-attention enables long-range dependencies to be captured.
- **State-of-the-Art Results:** Backbone of modern NLP models like BERT, GPT, T5, and more.

Applications

- **NLP Tasks:**
  - Text classification, sentiment analysis, translation, summarization, and question-answering.
- **Vision:**
  - Vision Transformers (ViT) for image classification and other computer vision tasks.
- **Audio and Speech:**
  - Transformers are used for speech recognition and audio generation.
- **Code Understanding:**
  - Models like Codex use Transformers to generate and interpret code.

Transformer-Based Models

- **BERT (Bidirectional Encoder Representations from Transformers):**
    - Focuses on understanding (NLP tasks like classification, QA).
- **GPT (Generative Pre-trained Transformer):**
    - Focuses on generating coherent text.
- **T5 (Text-to-Text Transfer Transformer):**
    - A unified framework for NLP tasks where everything is treated as a text-to-text problem.

How Generative AI Models use transformer-based architecture.

- Generative AI models, such as **GPT (Generative Pre-trained Transformer)**, **BERT**, **T5**, **DALL-E**, and others, leverage the **Transformer architecture** to generate text, images, code, and more. The key innovation of Transformers is their **attention mechanism**, which enables models to handle long-range dependencies and generate coherent, context-aware outputs.

Core Components of Generative AI Models Using Transformers

1. Self-Attention Mechanism

- Enables the model to focus on different parts of the input sequence.
- Helps the model learn relationships between tokens in the sequence, even when they are far apart.

2. Positional Encoding

- Adds sequence-order information to embeddings, ensuring the model understands the structure of input data.

3. Decoder-Only vs. Encoder-Decoder Architecture

- **Decoder-Only (e.g., GPT):**
  - Generates outputs token by token.
  - Uses causal (masked) self-attention to prevent attending to future tokens during generation.
  - Ideal for tasks like text generation, code generation, and conversational AI.
- **Encoder-Decoder (e.g., T5, DALL-E):**
  - Encodes an input sequence into a context vector and uses the decoder to generate outputs.
  - Suitable for tasks like translation, summarization, and image generation from text.

Steps in Generative AI Using Transformers

1. Pre-Training

- Models are trained on massive datasets using self-supervised learning.
- Example tasks:
  - **Causal Language Modeling (CLM):** Predict the next token in a sequence. Used in GPT.
  - **Masked Language Modeling (MLM):** Predict masked tokens in a sequence. Used in BERT.

2. Fine-Tuning

- Pre-trained models are fine-tuned on task-specific datasets for applications like:
  - Text summarization.
  - Dialogue generation.
  - Code generation (e.g., Codex).
  - Image generation (e.g., DALL-E).

3. Generation

- Models generate outputs using techniques like:
  - **Greedy Decoding:** Choose the token with the highest probability at each step.
  - **Beam Search:** Explore multiple output sequences simultaneously for the best result.
  - **Temperature Sampling:** Introduce randomness to the output by sampling from the probability distribution.

Popular Generative AI Models Using Transformers

1. GPT (Generative Pre-trained Transformer)
- **Architecture:** Decoder-only Transformer.
- **Functionality:** Text generation, conversation, story writing, and code generation.
- **Mechanism:**
  - Generates one token at a time using causal self-attention.
  - Example: Autocomplete for a sentence like "The weather is" to "The weather is sunny today."

2. BERT (Bidirectional Encoder Representations from Transformers)
- **Architecture:** Encoder-only Transformer.
- **Functionality:** Focuses on understanding input sequences rather than generation.
- **Generative Use:**
  - Combined with decoders to enable text generation (e.g., T5, BART).

3. T5 (Text-to-Text Transfer Transformer)
- **Architecture:** Encoder-Decoder Transformer.
- **Functionality:** Treats every NLP task as a text-to-text problem.
- **Example:**
  - Input: "Summarize: The cat sat on the mat. It was a sunny day."
  - Output: "The cat sat on the mat on a sunny day."

4. DALL-E
- **Architecture:** Transformer adapted for image generation.
- **Functionality:** Generates images from text descriptions.
- **Mechanism:**
  - Combines text embeddings with an autoregressive Transformer to create images pixel by pixel or patch by patch.

Generative AI Workflow Using Transformers

- **Input Representation:**
  - Tokenize and embed the input sequence (e.g., text or image).
  - Add positional encoding.
- **Attention-Based Learning:**
  - Use multi-head self-attention to understand relationships between tokens.
- **Decoding:**
  - For text generation, decode the output one token at a time, attending to the input and previously generated tokens.
- **Output Generation:**
  - Use a softmax layer to generate probabilities for the next token.
  - Convert the token IDs back to human-readable text or pixels.

Challenges

- **High Computational Costs:**
  - Requires significant computational resources for training and inference.
- **Memory Limitations:**
  - Self-attention has quadratic complexity with respect to sequence length.
- **Bias and Ethics:**
  - Can generate biased or harmful content if the training data is biased.

Real-World Examples of Generative AI with Transformers

- **ChatGPT:**
  - Builds conversational AI for interactive question-answering.
- **Google Bard:**
  - AI assistant for language-based tasks.
- **DALL-E 2:**
  - Generates high-quality images from text prompts.
- **GitHub Copilot:**
  - Assists developers by generating code snippets from comments.

Thank You…

Welcome to the world of Generative AI