[Week 11] LLM Foundations

ETMI5: Explain to Me in 5

In the first week of our course, we looked at the difference between two types of machine learning models: generative models, which LLMs are a part of, and discriminative models. Generative models are good at learning from data and creating new things. This week, we'll learn about how LLMs were developed by looking at the history of neural networks used in language processing. We start with the basics of Recurrent Neural Networks (RNNs) and move to more advanced architectures like sequence-to-sequence models, attention mechanisms, and transformers We'll also review some of the earlier language models that used transformers, like BERT and GPT. Finally, we'll talk about how the LLMs we use today were built on these earlier developments.

Generative vs Discriminative models

In the first week, we briefly covered the idea of Generative AI. It's essential to note that all machine learning models fall into one of two categories: generative or discriminative. LLMs belong to the generative category, meaning they learn text features and produce them for various applications. While we won't delve deeply into the mathematical intricacies, it's important to grasp the distinctions between generative and discriminative models to gain a general understanding of how LLMs operate:

Generative Models

Generative models try to understand how data is generated. They learn the patterns and structures in the data so they can create new similar data points.

For example, if you have a generative model for images of dogs, it learns what features and characteristics make up a dog (like fur, ears, and tails), and then it can generate new images of dogs that look realistic, even though they've never been seen before.

Discriminative Models

Discriminative models, on the other hand, are focused on making decisions or predictions based on the input they receive.

Using the same example of images of dogs, a discriminative model would look at an image and decide whether it contains a dog or not. It doesn't worry about how the data was generated; it's just concerned with making the right decision based on the input it's given.

Therefore, Generative models learn the underlying patterns in the data to create new samples, while discriminative models focus on making decisions or predictions based on the input data without worrying about how the data was generated.

Essentially, generative models create, while discriminative models classify or predict.

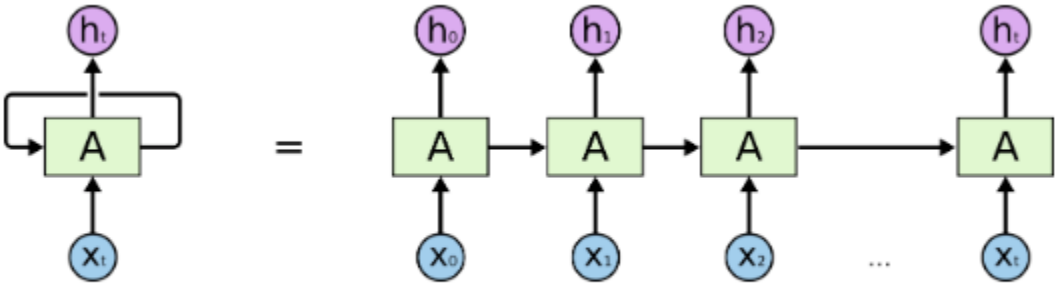
Neural Networks for Language

For several years, neural networks have been integral to machine learning. Among these, a prominent class of models heavily reliant on neural networks is referred to as deep learning models. The initial neural network type introduced for text generation was termed as a Recurrent Neural Network (RNN). Subsequent iterations with improvements emerged later, such as Long Short-Term Memory networks (LSTMs), Bidirectional LSTMs, and Gated Recurrent Units (GRUs). Now, let's explore how RNNs generate text.

Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) are a type of artificial neural network designed to handle sequential data by allowing information to persist through loops within the network architecture. Traditional neural networks lack the ability to retain information over time, which can be a major limitation when dealing with sequential data like text, audio, or time-series data.

The basic principle behind RNNs is that they have connections that form a directed cycle, allowing information to be passed from one step of the network to the next. This means that the output of the network at a particular time step depends not only on the current input but also on the previous inputs and the internal state of the network, which captures information from earlier time steps.



An unrolled recurrent neural network.

Screenshot 2024-02-23 at 10.24.38 AM.png

Image Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Here's a simplified explanation of how RNNs work:

1. **Input Processing**: At each time step t, the RNN receives an input x_t. This input could be a single element of a sequence (e.g., a word in a sentence) or a feature vector representing some aspect of the input data.
2. **State Update**: The input x_t is combined with the internal state h_{t-1} of the network from the previous time step to produce a new state h_t using a set of weighted

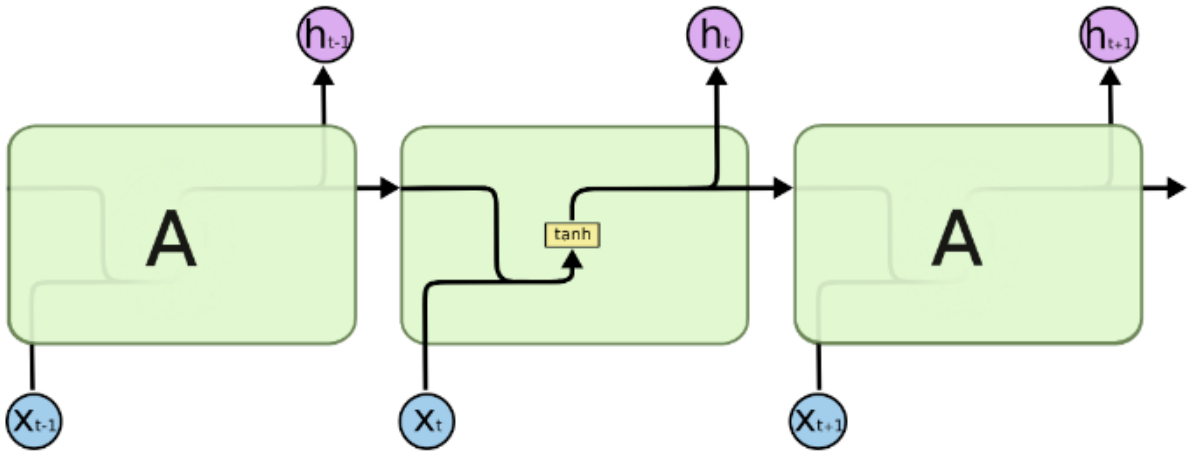
connections (parameters) within the network. This update process allows the network to retain information from previous time steps.

3. **Output Generation**: The current state h_t is used to generate an output y_t at the current time step. This output can be used for various tasks, such as classification, prediction, or sequence generation.
4. **Recurrent Connections**: The key feature of RNNs is the presence of recurrent connections, which allow information to flow through the network over time. These connections create a form of memory within the network, enabling it to capture dependencies and patterns in sequential data.

While RNNs are powerful models for handling sequential data, they can suffer from certain limitations, such as difficulties in learning long-range dependencies and vanishing/exploding gradient problems during training. To address these issues, more advanced variants of RNNs, such as Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs), have been developed. These architectures incorporate mechanisms for better handling long-term dependencies and mitigating gradient-related problems, leading to improved performance on a wide range of sequential data tasks.

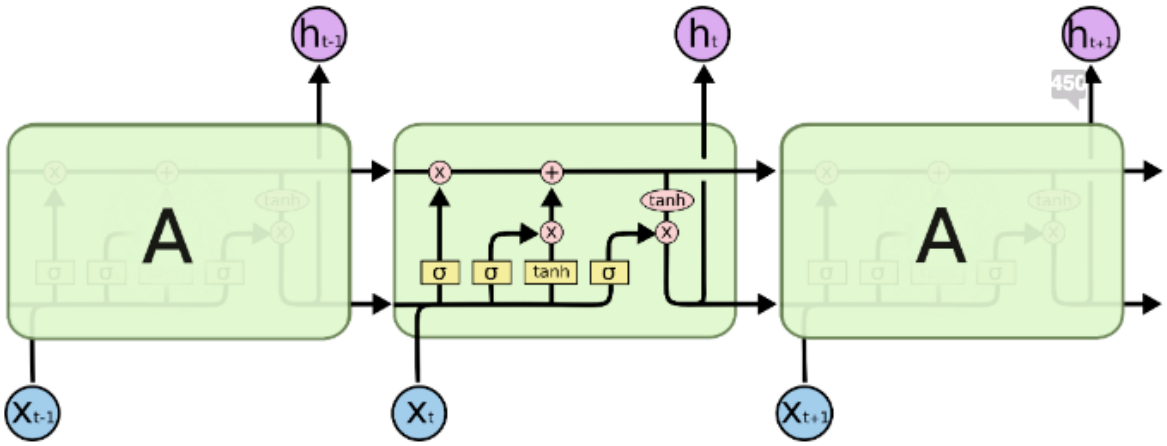
Long Short-Term Memory (LSTM)

LSTM networks are thus an enhanced version of RNNs designed to better handle sequences of data like text just like RNNs, but with the below improvements:



The repeating module in a standard RNN contains a single layer.

Screenshot 2024-02-23 at 10.32.53 AM.png



The repeating module in an LSTM contains four interacting layers.

Screenshot 2024-02-23 at 10.33.00 AM.png

Image Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

1. **Memory Cell**: LSTMs have a special memory cell that can store information over time.
2. **Gating Mechanism**: LSTMs use gates to control the flow of information into and out of the memory cell:
   - Input Gate: Decides how much new information to keep.
   - Forget Gate: Decides how much old information to forget.
   - Output Gate: Decides how much of the current cell state to output.
3. **Gradient Flow**: LSTMs help gradients flow better during training, which helps in learning from long sequences of data.
4. **Learning Long-Term Dependencies**: LSTMs are good at remembering important information from earlier in the sequence, making them useful for tasks where understanding context over long distances is crucial.

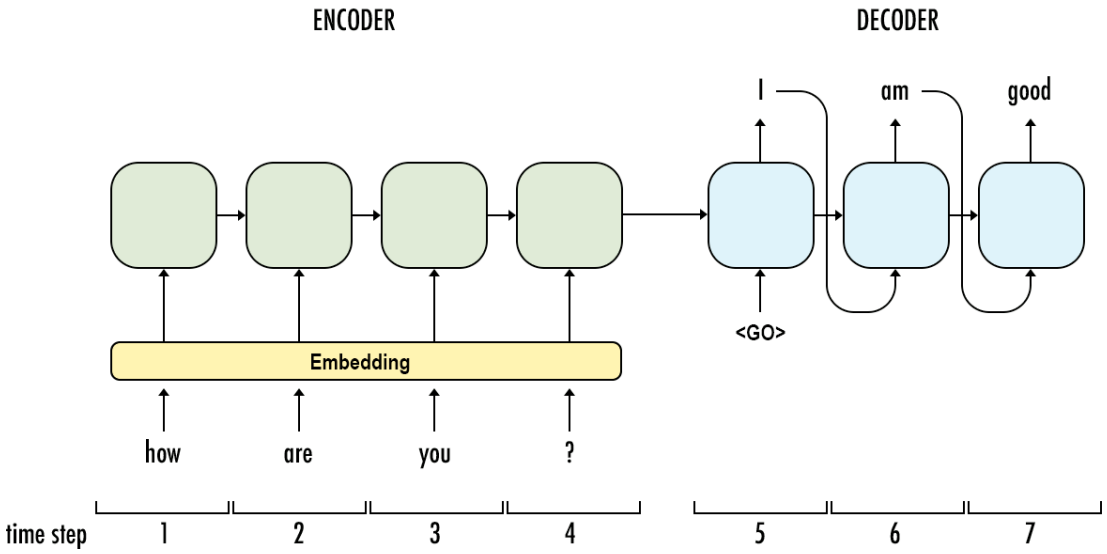
Therefore LSTMs are better at handling sequences by remembering important information and forgetting what's not needed, which makes them more effective than traditional RNNs for tasks like language processing.

Both RNNs and LSTMs (and their variants) are widely used for language modeling tasks, where the goal is to predict the next word in a sequence of words. They can learn the underlying structure of language and generate coherent text. However, they struggle to handle input sequences of variable lengths and generate output sequences of variable lengths because their fixed-size hidden states limit their ability to capture long-range dependencies and maintain context over time.

Sequence-to-Sequence (Seq2Seq) models

That's where Sequence-to-Sequence (Seq2Seq) models come in; they work by employing an encoder-decoder architecture, where the input sequence is encoded into a fixed-size representation (context vector) by the encoder, and then decoded into an output sequence by the decoder. This architecture allows Seq2Seq models to handle sequences of variable lengths and effectively capture the semantic meaning and structure of the input sequence while generating the corresponding output sequence. A simple Seq2Seq model is depicted below. Each unit in the Seq2Seq is still an RNN type of architecture.

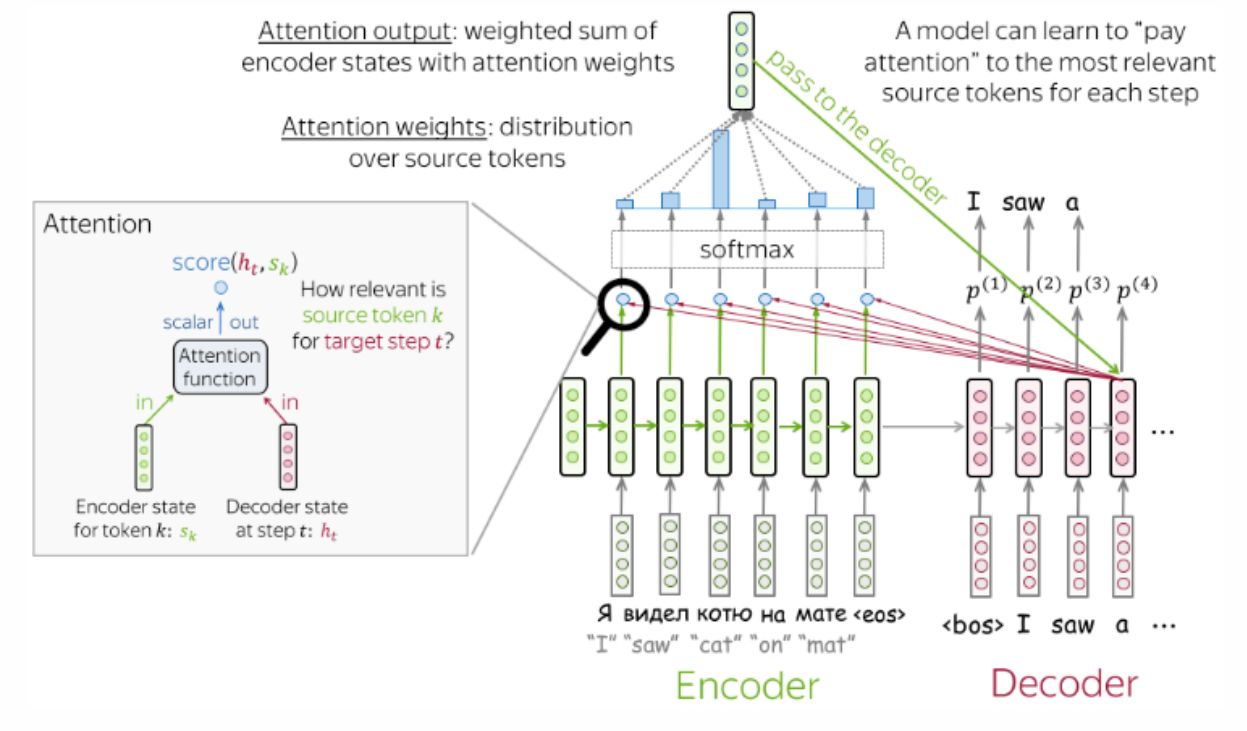
We won't dive too deep into the workings here for brevity, [this](https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/#:~:text=Sequence%20to%20Sequence%20(often%20abbreviated,Chatbots%2C%20Text%20Summarization%2C%20etc.) article is a great read for those interested:



s2s_11.png

Image Source: https://towardsdatascience.com/sequence-to-sequence-model-introduction-and-concepts-44d9b41cd42d

Seq2Seq models + Attention



Screenshot 2024-02-24 at 2.19.47 PM.png

Image Source: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

The problem with traditional Seq2Seq models lies in their inability to effectively handle long input sequences, especially when generating output sequences of variable lengths. In standard Seq2Seq models, a fixed-length context vector is used to summarize the entire input sequence, which can lead to information loss, particularly for long sequences. Additionally, when generating output sequences, the decoder may struggle to focus on relevant parts of the input sequence, resulting in suboptimal translations or predictions.

To address these issues, attention mechanisms were introduced. Attention mechanisms allow Seq2Seq models to dynamically focus on different parts of the input sequence during the decoding process.

Here's how attention works:

1. **Encoder Representation**: First, the input sequence is processed by an encoder. The encoder converts each word or element of the input sequence into a hidden state. These hidden states represent different parts of the input sequence and contain information about the sequence's content and structure.
2. **Calculating Attention Weights**: During decoding, the decoder needs to decide which parts of the input sequence to focus on. To do this, it calculates attention

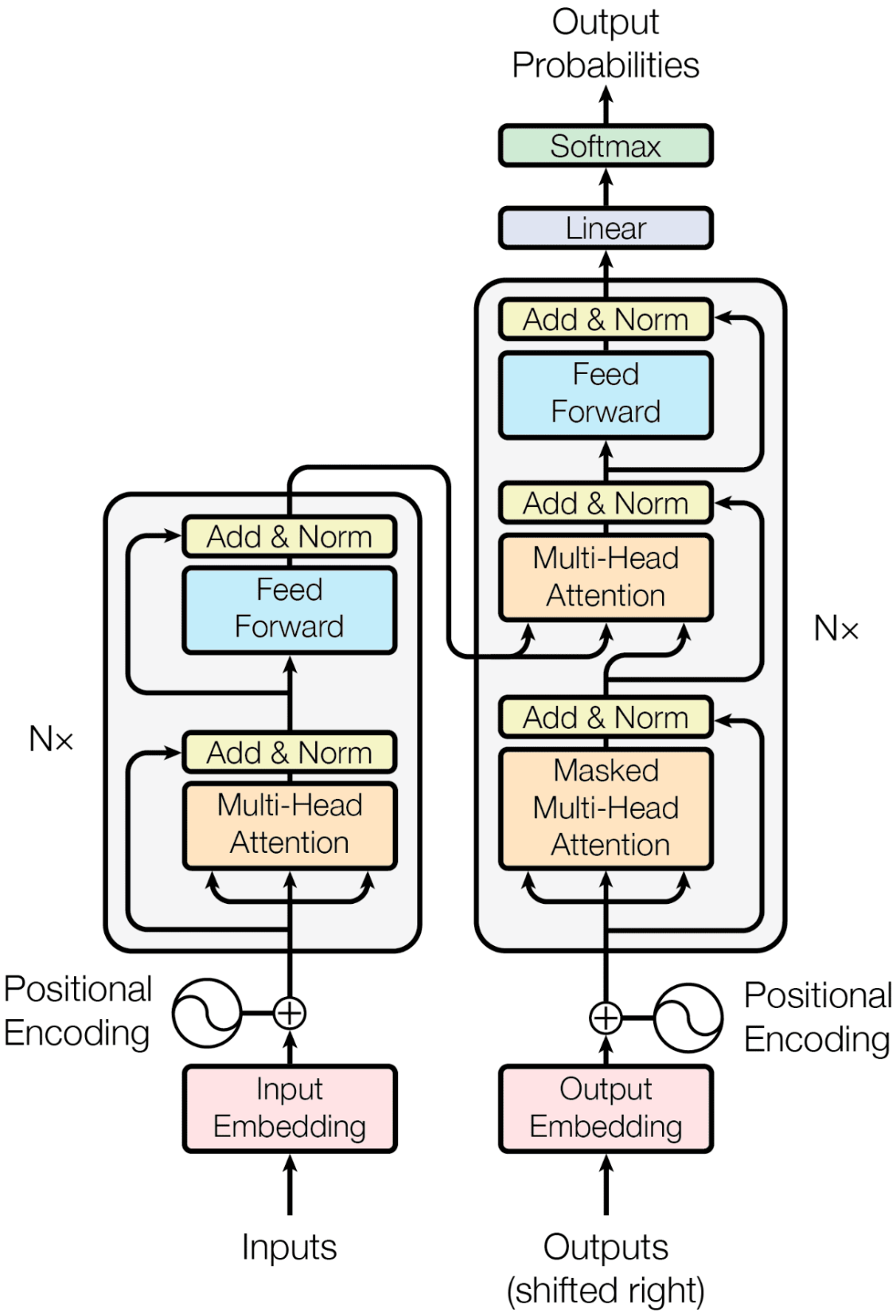
weights. These weights indicate the relevance or importance of each encoder hidden state to the current decoding step. Essentially, the model is trying to determine which parts of the input sequence are most relevant for generating the next output token.

3. **Softmax Normalization**: After calculating the attention weights, the model normalizes them using a softmax function. This ensures that the attention weights sum up to one, effectively turning them into a probability distribution. By doing this, the model can ensure that it allocates its attention appropriately across different parts of the input sequence.
4. **Weighted Sum**: With the attention weights calculated and normalized, the model then takes a weighted sum of the encoder hidden states. Essentially, it combines information from different parts of the input sequence based on their importance or relevance as determined by the attention weights. This weighted sum represents the "attended" information from the input sequence, focusing on the parts that are most relevant for the current decoding step.
5. **Combining Context with Decoder State**: Finally, the context vector obtained from the weighted sum is combined with the current state of the decoder. This combined representation contains information from both the input sequence (through the context vector) and the decoder's previous state. It serves as the basis for generating the output of the decoder for the current decoding step.
6. **Repeating for Each Decoding Step**: Steps 2 to 5 are repeated for each decoding step until the end-of-sequence token is generated or a maximum length is reached. At each step, the attention mechanism helps the model decide where to focus its attention in the input sequence, enabling it to generate accurate and contextually relevant output sequences.

Transformer Models

The problem with Seq2Seq models with attention lies in their computational inefficiency and inability to capture dependencies effectively across long sequences. While attention mechanisms significantly improve the model's ability to focus on relevant parts of the input sequence during decoding, they also introduce computational overhead due to the need to compute attention weights for each decoder step. Additionally, like we mentioned before, traditional Seq2Seq models with attention still rely on RNN or LSTM networks, which have limitations in capturing long-range dependencies.

The Transformer model was introduced to address these limitations and improve the efficiency and effectiveness of sequence-to-sequence tasks. Here's how the Transformer model solves the problems of Seq2Seq models with attention:


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

transformer_11

Image Source: https://arxiv.org/pdf/1706.03762.pdf

1. **Self-Attention Mechanism**: Instead of relying solely on attention mechanisms between the encoder and decoder, the Transformer model introduces a self-attention mechanism. This mechanism allows each position in the input sequence to attend to all other positions, capturing dependencies across the entire input sequence simultaneously. Self-attention enables the model to capture long-range dependencies more effectively compared to traditional Seq2Seq models with attention.
2. **Parallelization**: The Transformer model relies on self-attention layers that can be computed in parallel for each position in the input sequence. This parallelization greatly improves the model's computational efficiency compared to traditional Seq2Seq models with recurrent layers, which process sequences sequentially. As a result, the Transformer model can process sequences much faster, making it more suitable for handling long sequences and large-scale datasets.
3. **Positional Encoding**: Since the Transformer model does not use recurrent layers, it lacks inherent information about the order of elements in the input sequence. To address this, positional encoding is added to the input embeddings to provide information about the position of each element in the sequence. Positional encoding allows the model to distinguish between elements based on their position, ensuring that the model can effectively process sequences with ordered elements.
4. **Transformer Architecture**: The Transformer model consists of an encoder-decoder architecture, similar to traditional Seq2Seq models. However, it replaces recurrent layers with self-attention layers, which enables the model to capture dependencies across long sequences more efficiently. Additionally, the Transformer architecture allows for greater flexibility and scalability, making it easier to train and deploy on various tasks and datasets.

In summary, the Transformer model addresses the limitations of Seq2Seq models with attention by introducing self-attention mechanisms, parallelization, positional encoding, and a flexible architecture. These advancements improve the model's ability to capture long-range dependencies, process sequences efficiently, and achieve state-of-the-art performance on various sequence-to-sequence tasks.

Older Language Models

Although LLMs have gained significant attention recently, especially with models like GPT from OpenAI, it's important to recognize that the groundwork for this architecture was laid by earlier models such as BERT, GPT (older versions) and T5 explained below.

LLMs like BERT (Bidirectional Encoder Representations from Transformers), GPT (Generative Pre-trained Transformer), and T5 (Text-To-Text Transfer Transformer) build on top of the concepts introduced by the Transformer model (described in the previous sections) using the following steps:

1. **Pre-training and Fine-Tuning**: These models utilize a pre-training and fine-tuning approach. During pre-training, the model is trained on large-scale corpora using unsupervised learning objectives, such as masked language modeling (BERT), autoregressive language modeling (GPT), or text-to-text pre-training (T5). This pre-training phase allows the model to learn rich representations of language and general knowledge from large amounts of text data. After pre-training, the model can be fine-tuned on specific downstream tasks with labeled data, enabling it to adapt its learned representations to perform various NLP tasks such as text classification, question answering, and machine translation.
2. **Bidirectional Context**: BERT introduced bidirectional context modeling by utilizing a masked language modeling objective. Instead of processing text in a left-to-right or right-to-left manner, BERT is able to consider context from both directions by masking some of the input tokens and predicting them based on the surrounding context. This bidirectional context modeling enables BERT to capture deeper semantic relationships and dependencies within text, leading to improved performance on a wide range of NLP tasks.
3. **Autoregressive Generation**: GPT models leverage autoregressive generation, where the model predicts the next token in a sequence based on the previously generated tokens. This approach allows GPT models to generate coherent and contextually relevant text by considering the entire history of the generated sequence. GPT models are particularly effective for tasks that involve generating natural language, such as text generation, dialogue generation, and summarization.
4. **Text-to-Text Approach**: T5 introduces a unified text-to-text framework, where all NLP tasks are framed as text-to-text mapping problems. This approach unifies various NLP tasks, such as translation, classification, summarization, and question answering, under a single framework, simplifying the training and deployment process. T5 achieves this by representing both the input and output of each task as textual strings, enabling the model to learn a single mapping function that can be applied across different tasks.
5. **Large-Scale Training**: These models are trained on large-scale datasets containing billions of tokens, leveraging massive computational resources and distributed training techniques. By training on extensive data and with powerful hardware, these models can capture rich linguistic patterns and semantic relationships, leading to significant improvements in performance across a wide range of NLP tasks.

Large Language Models

The latest Llama such as Llama and ChatGPT represent significant advancements over earlier models like BERT and GPT in several key ways:

1. **Task Specialization**: While earlier LLMs like BERT and GPT were designed to perform a wide range of NLP tasks, including text classification, language generation, and question answering, newer models like Llama and ChatGPT are more specialized. For example, Llama is specifically tailored for multimodal tasks, such as image captioning and visual question answering, while ChatGPT is optimized for conversational applications, such as dialogue generation and chatbots.

2. **Multimodal Capabilities**: Llama and other recent LLMs integrate multimodal capabilities, allowing them to process and generate text in conjunction with other modalities such as images, audio, and video. This enables LLMs to perform tasks that require understanding and generating content across multiple modalities, opening up new possibilities for applications like image captioning, video summarization, and multimodal dialogue systems.
3. **Improved Efficiency**: Recent advancements in LLM architecture and training methodologies have led to improvements in efficiency, allowing models like Llama and ChatGPT to achieve comparable performance to their predecessors with fewer parameters and computational resources. This increased efficiency makes it more practical to deploy these models in real-world applications and reduces the environmental impact associated with training large models.
4. **Fine-Tuning and Transfer Learning**: LLMs like ChatGPT are often fine-tuned on specific datasets or tasks to further improve performance in targeted domains. By fine-tuning on domain-specific data, these models can adapt their pre-trained knowledge to better suit the requirements of particular applications, leading to enhanced performance and generalization.
5. **Interactive and Dynamic Responses**: ChatGPT and similar conversational models are designed to generate interactive and dynamic responses in natural language conversations. These models leverage context from previous turns in the conversation to generate more coherent and contextually relevant responses, making them more suitable for human-like interaction in chatbot applications and dialogue systems.

Read/Watch These Resources (Optional)

1. Understanding LSTM Networks: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
2. Sequence to Sequence (seq2seq) and Attention: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
3. Sequence to Sequence models: https://www.youtube.com/watch?v=kklo05So99U
4. How Attention works in Deep Learning: understanding the attention mechanism in sequence models**:** https://theaisummer.com/attention/
5. Intro to LLMs:
    1. https://www.youtube.com/watch?v=zjkBMFhNj_g&t=1845s
    2. https://www.youtube.com/watch?v=zizonToFXDs
6. Transformers: https://www.youtube.com/watch?v=wl3mbqOtlmM

Read These Papers (Optional)

1. https://arxiv.org/abs/1706.03762
2. https://arxiv.org/abs/2005.14165
3. https://arxiv.org/abs/1910.10683