From RNNs to Transformers: A Gentle Review on Modern NLP Models

Tooba Rahimnia
Department of Electrical and Computer Engineering
McGill University
tooba.rahimnia@mail.mcgill.ca

Abstract

This paper provides an overview of key models in natural language processing (NLP), focusing on their architectures, applications, and limitations. We discuss foundational models such as GPT, BERT, and T5, highlighting their contributions to the field and examining their strengths and challenges. Additionally, we explore the transition of transformer models from NLP to computer vision, exemplified by the Vision Transformer (ViT), showcasing the versatility of self-attention mechanisms across domains. This guide aims to offer a concise, accessible resource for those new to NLP, providing a solid foundation for understanding the current landscape of foundational models.

1 Introduction

Early approaches to natural language processing (NLP) relied heavily on rule-based systems and statistical methods, such as Statistical Machine Translation (SMT), which utilized extensive datasets and probabilistic techniques to address language tasks. A major transformation occurred in 2014 with the emergence of Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, which became instrumental in processing sequential data, notably in machine translation.

The field witnessed a paradigm shift in 2017 with the introduction of the Attention Mechanism, as presented by Vaswani et al. in their seminal work, "Attention Is All You Need" [28]. This innovation revolutionized the handling of sequential data by enabling models to capture long-range dependencies more effectively, overcoming the inherent limitations of RNNs and LSTMs due to their sequential processing.

The rapid evolution of NLP models and their diverse applications has significantly reshaped the landscape of artificial intelligence. These models are not only advancing capabilities in language understanding and generation but also laying the groundwork for transformative AI applications in fields such as healthcare, robotics, and beyond. This paper aims to provide an in-depth overview of contemporary NLP models, examining their impact, limitations, and critical role in modern NLP tasks and beyond.

2 Historical Context

Advancements in natural language processing (NLP) date back to the 1950s, when researchers at IBM and Georgetown University developed a system capable of automatically translating phrases from Russian to English [1]. A decade later, MIT researcher Joseph Weizenbaum (1966) made a significant breakthrough by introducing Eliza, the world's first chatbot [2]. This pioneering program employed pattern recognition to simulate conversations, transforming user input into questions and generating

Preprint. Under review.

responses based on predefined rules. While Eliza was far from perfect, it laid the groundwork for the development of more advanced language models.

2.1 Hidden Markov Model

Natural Language Processing (NLP) relies heavily on statistical modeling. Statistical NLP leverages statistical inferences drawn from data generated by unknown probability distributions [3]. Among the various approaches in machine learning for NLP, the Markov model is particularly significant [4].

In the early 1970s, Leonard E. Baum and colleagues introduced the Hidden Markov Model (HMM) in the field of artificial intelligence [5]. HMMs utilize probabilities to determine word roles (e.g., noun, verb) within a sentence. They are particularly useful when the underlying process generating observations is unknown, hence the name "Hidden Markov Model". HMMs predict future observations or classify sequences based on the hidden processes that generate the data.

A HMM consists of two types of variables: hidden states and observations. The hidden states $S = s_1, s_2, ..., s_N$ are the variables that generate the observed data, but they are not directly observable. The observations $O = o_1, o_2, ..., o_T$ are the variables that are measured and observed.

The relationship between the hidden states and the observations is modeled using two sets of probabilities: the transition probabilities and the emission probabilities. The transition probabilities $P(s_i \mid s_{i-1})$ describe the probability of transitioning from one hidden state to another. The emission probabilities $P(o_j \mid s_i)$ describe the probability of observing an output o_j given a hidden state s_i.

The shift to statistical approaches enhanced language processing flexibility and context sensitivity but required substantial computational resources and data. This transition also introduced challenges, driving future advancements in language modeling.

2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) were introduced in the early 1980s with the foundational work by John Hopfield, who developed the Hopfield Network in 1982 [6]. RNNs are primarily used for pattern detection in sequential data, which can include handwriting, text, genomes, or numerical time series [7, 12]. RNNs have also been widely used in language modeling, text generation, speech recognition, and generating image descriptions or video tagging [13].

A recurrent neural network architecture is structured with loops, meaning it takes into account both the input at the current time step and the hidden state from the previous time step [14]. This introduces a temporal dependency, making the hidden state at time t dependent on all preceding states $\mathbf{h}_{t-1}$, $\mathbf{h}_{t-2}$, ...$\mathbf{h}_0$.

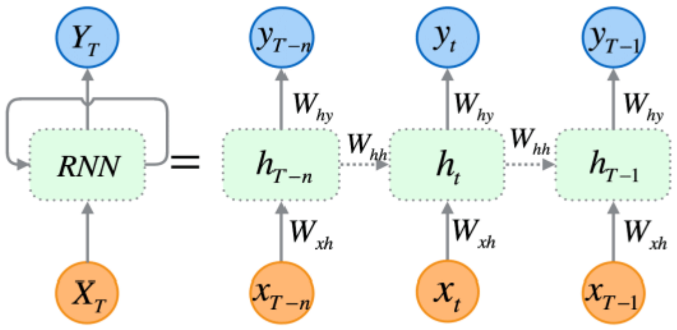

Figure 1: RNN example: recursive description for RNNs (left), the corresponding extended RNN model in a time sequential manner (right). Source [36].

Mathematically, $\mathbf{h}_t$ and $\mathbf{z}_t$ can be written as

$$\mathbf{h}_t = \phi(W_{xh}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + b_h) \tag{1}$$

$$\mathbf{z}_t = \phi(W_{hz}\mathbf{h}_t + b_z) \tag{2}$$

where, $\mathbf{h}_t$ represents the hidden state at time step t, $\mathbf{x}_t$ denotes the input at time step t, and $\mathbf{z}_t$ is the output at time step t. The function ϕ typically applies a non-linearity, such as a hyperbolic tangent (tanh), sigmoid, or softmax.

For simplicity, the RNN model shown in the figure above has a single hidden layer, though it can be extended to multiple layers. Additionally, since sequential data can vary in length, we assume the parameters remain consistent across all time steps. This assumption simplifies gradient computation, as varying parameters would complicate the process.

3 Backpropagation Through Time (BPTT)

Backpropagation Through Time (BPTT) is an extension of the standard backpropagation algorithm used to train RNNs [15]. The recurrent nature of RNNs introduces dependencies over time, meaning that when you compute the loss at a certain time step, it is influenced by all previous time steps. This results in a need to propagate errors backward through multiple time steps, making the optimization process more complex.

BPTT addresses this challenge by unrolling the RNN over time. It essentially converts the recurrent neural network into a very deep neural network by unfolding it for each time step in the sequence. By doing this, we can apply backpropagation as if the network were feedforward, but over multiple time steps. At each time step t, the network computes the hidden state $\mathbf{h}_t$ and the output based on the current input and the previous hidden state. The loss accumulated over time is

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{t=1}^{T} \ell(\hat{\mathbf{y}}_t, \mathbf{y}_t) \tag{3}$$

and $\ell(\hat{\mathbf{y}}_t, \mathbf{y}_t)$ is the loss at time step t. Once the forward pass is complete, gradients are backpropagated through the unfolded network. The objective with BPTT is to find the optimal weights and biases in each update using gradient descent. Three categories of weights to be updated are: the weights connecting inputs $\mathbf{W}_{xh}$, the weights through the hidden states $\mathbf{W}_{hh}$, and the output weights $\mathbf{W}_{hz}$. With the chain rule which is also used in normal backpropagation we get to the result for $\mathbf{W}_{hz}$ shown in Eq. 4

$$\frac{\partial \mathcal{L}}{\partial W_{hz}} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{t-i}} \frac{\partial \mathbf{y}_{t-i}}{\partial W_{hz}} \tag{4}$$

and for the partial derivative with respect to $\mathbf{W}_{hh}$ we can get the equation

$$\frac{\partial \mathcal{L}}{\partial W_{hh}} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{t-i}} \frac{\partial \mathbf{y}_{t-i}}{\partial \mathbf{h}_{t-i}} \left(\prod_{j=t-i+1}^{t} \frac{\partial \mathbf{h}_{t-j+1}}{\partial \mathbf{h}_{t-j}} \right) \frac{\partial \mathbf{h}_{t-i-1}}{\partial W_{hh}} \tag{5}$$

where $\mathbf{h}_{t-i}$ represents the hidden state at time step $t-i$. The symbol $\prod$ is used to multiply the gradients of hidden states through time. For the partial derivative with respect to $\mathbf{W}_{xh}$ the result we get is shown in the equation 6

$$\frac{\partial \mathcal{L}}{\partial W_{xh}} = \sum_{i=0}^{t} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{t-i}} \frac{\partial \mathbf{y}_{t-i}}{\partial \mathbf{h}_{t-i}} \left(\prod_{j=t-i+1}^{t} \frac{\partial \mathbf{h}_{t-j+1}}{\partial \mathbf{h}_{t-j}} \right) \frac{\partial \mathbf{h}_{t-i-1}}{\partial W_{xh}} \tag{6}$$

Despite various improvements, RNNs still suffer from the vanishing and exploding gradient problems. These issues arise because the gradients used to update the model weights (as shown in Equations 4, 5, and 6) can either become excessively large (exploding) or diminish to nearly zero (vanishing) when propagated through multiple layers or time steps. For example, in the case of vanishing gradient, as shown in Eq. 6, the term $\frac{\partial \mathbf{h}_{t-j+1}}{\partial \mathbf{h}_{t-j}}$ indicates matrix multiplication over the time sequence, leading to the compounding of gradients. As RNNs backpropagate through long sequences, the gradient values decrease layer by layer, eventually becoming too small to facilitate learning. When gradients vanish, updates are minimal, making it difficult for the model to capture long-term dependencies. Therefore, as that gap grows, RNNs become unable to learn to connect the information.

4 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a specialized form of Recurrent Neural Network (RNN) that address the limitations of standard RNNs, specifically the vanishing gradient problem, which impairs the learning of long-term dependencies [9, 10]. LSTM networks achieve this by incorporating a memory cell that acts as a long-term storage mechanism, maintaining relevant information across different time steps during sequence processing. This memory cell allows LSTMs to effectively preserve and update information over time, enhancing the network's ability to learn relationships in sequential data.

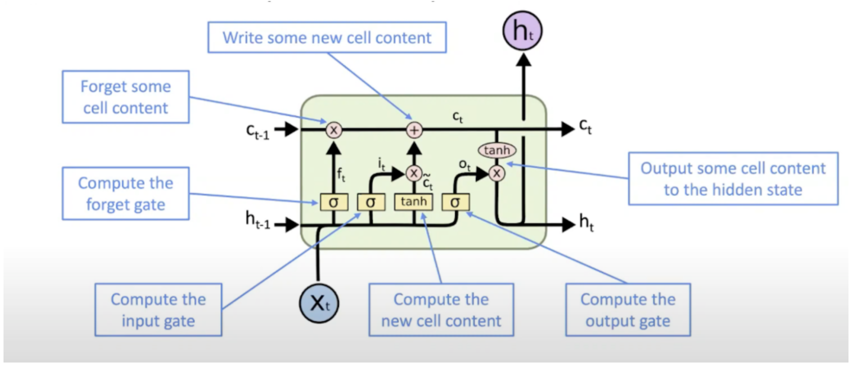

Figure 2: Visual representation of the LSTM (Long Short-Term Memory) architecture. Source [37].

The architecture of LSTM networks includes three gates—input, forget, and output—that regulate the flow of information into, within, and out of the memory cell. The forget gate selectively removes information from the memory cell by deciding which parts of the previous state are no longer relevant. The computations for this gate is shown in Eq. 7

$$\mathbf{F}_t = \sigma\left(\mathbf{x}_t W_{x_f} + \mathbf{h}_{t-1} W_{h_f} + b_f\right) \tag{7}$$

The input gate controls how much new information should be added to the cell state. It consists of a sigmoid function, which decides which values will be updated. There is candidate memory that contains a tanh function, which creates a new candidate value to be added to the cell state. The combination of input gate $\mathbf{I}_t$ and candidate memory $\tilde{\mathbf{C}}_t \in \mathbb{R}^{n \times h}$ determines what new information gets stored in the memory cell.

$$\mathbf{I}_t = \sigma\left(\mathbf{x}_t W_{x_i} + \mathbf{h}_{t-1} W_{h_i} + b_i\right) \tag{8}$$

$$\tilde{\mathbf{C}}_t = \tanh\left(\mathbf{x}_t W_{xc} + \mathbf{h}_{t-1} W_{hc} + b_c\right) \tag{9}$$

To regulate how much of the previous memory content $\mathbf{C}_{t-1} \in \mathbb{R}^{n \times h}$ is retained when updating to the new memory content $\mathbf{C}_t$, the previous memory is incorporated into the computation alongside the gating mechanisms, as shown in Eq. 10. Here, $\odot$ represents element-wise multiplication, allowing the gates to selectively control the flow and retention of information.

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \circ \tilde{\mathbf{C}}_t \tag{10}$$

The output gate determines which parts of the memory cell's content are passed on as the output of the current time step, thereby influencing the hidden state $\mathbf{h}_t \in \mathbb{R}^{n \times h}$ used in subsequent layers or time steps.

$$\mathbf{O}_t = \sigma\left(\mathbf{x}_t W_{x_o} + \mathbf{h}_{t-1} W_{h_o} + b_o\right) \tag{11}$$

$$\mathbf{h}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t) \tag{12}$$

The equations above use W_{x_i}, W_{x_f}, and $W_{x_o} \in \mathbb{R}^{d\times h}$ and W_{h_i}, W_{h_f}, and $W_{h_o} \in \mathbb{R}^{h\times h}$ as weight matrices while b_i, b_f, and $b_o \in \mathbb{R}^{1\times h}$ are their respective biases. Further, they use the sigmoid activation function σ to transform the output into (0, 1), resulting in a vector with entries in (0, 1). And with the tanh function we ensure that each element of $\tilde{\mathbf{C}}_t$ and $\mathbf{C}_t$ is $\in (-1, 1)$.

5 Bidirectional Recurrent Neural Networks (BRNNs)

When we use information only from the past time steps, we are using a unidirectional RNN. While effective for certain tasks, unidirectional RNNs struggle with tasks requiring knowledge of future context. To address this limitation, we make use of bidirectional recurrent neural networks (BRNNs) which capture both past and future contexts and can enhance prediction accuracy [19].

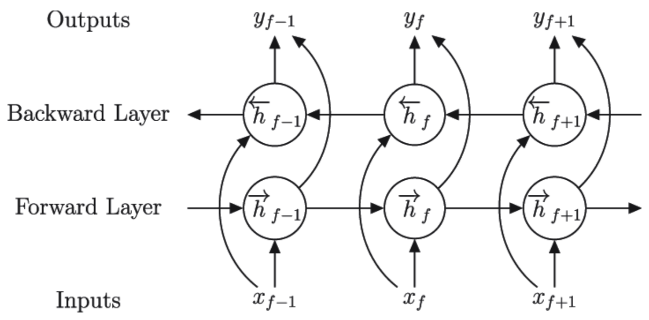

Figure 3: BRNN schema.

BRNNs achieve this by employing two separate sets of hidden layers, one that processes the sequence forward $\overrightarrow{\mathbf{h}_t} \in \mathbb{R}^{n\times h}$ and another that processes it backward $\overleftarrow{\mathbf{h}}_t \in \mathbb{R}^{n\times h}$.

$$\overrightarrow{\mathbf{h}}_t = \phi(\mathbf{x}_t W_{xh}^{(f)} + \overrightarrow{\mathbf{h}}_{t-1} W_{hh}^{(f)} + b_h^{(f)}) \tag{13}$$

$$\overleftarrow{\mathbf{h}}_t = \phi(\mathbf{x}_t W_{xh}^{(b)} + \overleftarrow{\mathbf{h}}_{t+1} W_{hh}^{(b)} + b_h^{(b)}) \tag{14}$$

For that, we have similar weight matrices as in definitions before but now they are seperated into two sets. One set of weight matrices is for the forward hidden states $W_{xh}^{(f)} \in \mathbb{R}^{d\times h}$ and $W_{hh}^{(f)} \in \mathbb{R}^{h\times h}$ while the other one is for the backward hidden states $W_{xh}^{(b)} \in \mathbb{R}^{d\times h}$ and $W_{hh}^{(b)} \in \mathbb{R}^{h\times h}$. They also have their respective biases $b_h^{(f)} \in \mathbb{R}^{1\times h}$ and $b_h^{(b)} \in \mathbb{R}^{1\times h}$. The final output $\mathbf{y}_t \in \mathbb{R}^{n\times o}$ with o at time step t is a combination (often concatenation or a weighted sum) of both the forward and backward hidden states.

$$\mathbf{y}_t = \phi\left(\left[\overrightarrow{\mathbf{h}}_t \parallel \overleftarrow{\mathbf{h}}_t\right] W_{ho} + b_o\right) \tag{15}$$

The two hidden state directions can have different number of hidden units.

6 Sequence-to-Sequence Models (Seq2Seq)

let's imagine we are translating a sentence from English to Mandarin. The input sentence has five words and the output sentence has seven words. The training of such task cannot be done with a regular LSTM as it cannot map each word from English to exactly one word in Mandarin.

We can use sequence to sequence models (Seq2Seq) to address such problems. Seq2Seq was first introduced in 2014 by Google [38] and it aims to map a fixed-size input to a fixed-size output where the size of input and output could be equal but not necessarily. The common use cases of Seq2Seq models are in machine translation [21], speech recognition [22], and video captioning [23].

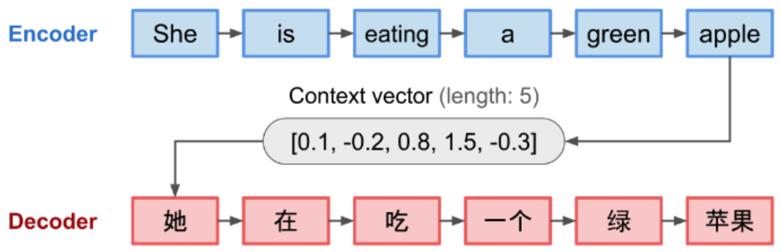

Figure 4: An example of a Seq2Seq model in machine translation. Source unknown.

The model consists of three parts: encoder, context vector and decoder. Both encoder and decoder stack several recurrent units (e.g. LSTM or GRU for optimal performance) where they encode the input into a state and decode the state to an output respectively. The encoder processes the input sequence $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$ and generates a context vector c, which summarizes the input sequence. The hidden state $\mathbf{h}_t$ at each time step in the encoder is calculated as

$$\mathbf{h}_t = \phi(W_{xh}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1}) \tag{16}$$

where ϕ is a non-linear activation function. The context vector $\mathbf{c}$ is typically the final hidden state $\mathbf{h}_t$ of the encoder, and it aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions. Context vector acts as the initial hidden state of the decoder part of the model. In the decoder, at each time step t, the hidden state $\mathbf{h}_t$ and the output $\hat{\mathbf{y}}_t$ are computed as shown in Eq 17 and Eq. 18. The output is usually calculated often via a softmax function to get a probability distribution over possible output tokens.

$$\mathbf{h}_t = \phi(W_{hh}\mathbf{h}_{t-1}) \tag{17}$$

$$\hat{\mathbf{y}}_t = \text{softmax}(W_s\mathbf{h}_t) \tag{18}$$

This architecture has two main limitations, both linked to sequence length. First, it has limited memory, with the context vector (final LSTM hidden state) responsible for capturing the entire sentence, typically containing only a few hundred units. This fixed size makes it difficult to maintain high-quality representations when overloaded with information. On the other hand, as the network deepens, training becomes more challenging. For RNNs, longer sequences result in deeper networks along the time dimension, leading to vanishing gradient issues. Although architectures like LSTM are designed to mitigate this, the problem persists. Approaches such as Attention mechanisms have been introduced to address these challenges, which we will discuss further in the next section.

7 The Attention Mechanism

At the core of human cognition is the concept of attention, a mechanism that allows us to focus on certain elements of our environment while filtering out irrelevant stimuli. This idea has influenced the creation of the attention mechanism in fields like image recognition and natural language processing [24]. In deep learning, the attention mechanism enables models to replicate this human ability by concentrating on the most pertinent parts of the input data.

At a high level, the attention mechanism assigns a weight of importance to each input state, reflecting how closely it is linked to the corresponding output state. Essentially, for each cell in the decoder, the model draws from all encoder states, with their contributions weighted by relevance. This method helps solve the bottleneck issue in Seq2Seq models, discussed earlier, by ensuring that every encoder state is taken into account during each decoding step.

The attention mechanism, introduced by Bahdanau et al. [25], was designed to overcome the challenge of preserving information from lengthy source sentences in neural machine translation (NMT). In contrast to traditional methods that rely on a fixed context vector derived from the encoder's final hidden state, attention creates a dynamic link between the context vector and the entire source input [24].

Consider the architecture proposed by [25]: given a source sequence $\mathbf{x}$ of length n, the mechanism's goal is to predict a target sequence $\mathbf{y}$ of length m, with the model selectively focusing on the relevant parts of the source sequence at each output step. The encoder is a bidirectional RNN (BRNN) that includes both a forward hidden state $\overrightarrow{\mathbf{h}}_i$ and a backward hidden state $\overleftarrow{\mathbf{h}}_i$. The combination of these two hidden states forms the encoder state.

$$\mathbf{h}_i = \left[\overrightarrow{\mathbf{h}}_i \parallel \overleftarrow{\mathbf{h}}_i\right], i = 1, 2, ..., n \tag{19}$$

The decoder's hidden state at position t, denoted as $\mathbf{s}_t = \phi(\mathbf{s}_{t-1}, \mathbf{y}_{t-1}, \mathbf{c}_t)$, is computed based on the previous hidden state, the previous output word, and the current context vector. Here, the context vector $\mathbf{c}_t$ is obtained by summing the encoder's hidden states for the input sequence, weighted according to alignment scores for each position.

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i}\mathbf{h}_i \tag{20}$$

Each alignment score $\alpha_{t,i}$, which reflects the connection between the input at position i and the output at position t, $(\mathbf{y}_t, \mathbf{x}_i)$, indicates the extent to which each source hidden state influences the generation of the output. These scores are then normalized via a softmax function, which transforms the raw alignment scores into probabilities that sum to one, ensuring the weights are properly scaled for each output step.

$$\alpha_{t,i} = \frac{\exp(\text{score}(\mathbf{s}_{t-1}, \mathbf{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\mathbf{s}_{t-1}, \mathbf{h}_{i'}))} \tag{21}$$

In Bahdanau's paper, the alignment score is modeled using a feed-forward network with a single hidden layer. This network is trained together with the rest of the model, enabling the alignment mechanism to evolve and improve as the model learns [24]. An example of this process is shown in Fig. 5, where white represents a strong correlation between words, and black indicates a weak correlation.

When working with large text corpora, RNNs with attention mechanisms face limitations. The sequential nature of these models leads to longer processing times as the input size increases. This necessitates the development of a model capable of handling inputs in parallel. This demand led to the creation of the Transformer model, which will be explored in detail in the next section.

8 Transformers

To overcome the challenge of sequential computation, transformers [28] process inputs in parallel. They are composed of encoders and decoders, each equipped with built-in attention mechanisms. The transformer model utilizes a particular form of attention called self-attention, which helps speed up the sequence-to-sequence transformation process [39].

8.1 Encoder and Decoder stacks

Transformers consist of six encoders and six decoders, as described in [28]. Each encoder includes two primary sub-layers: a self-attention mechanism and a feed-forward neural network, with the input first passing through the self-attention layer. The decoder follows a similar structure but also incorporates an additional encoder-decoder attention layer, which helps it focus on the most relevant

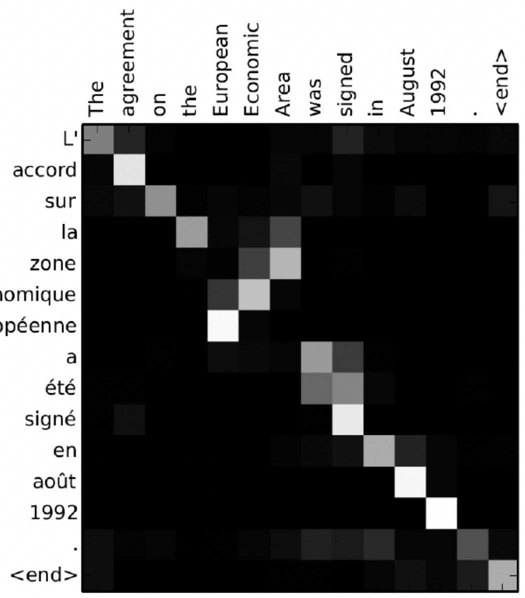

Figure 5: Alignment matrix of "L'accord sur l'Espace économique européen a été signé en août 1992" (French) and its English translation "The agreement on the European Economic Area was signed in August 1992". Source [25].

parts of the input sequence. A residual connection followed by a normalization layer is applied to all sub-layers of both the encoder and decoder [28].

The process begins by converting each input word into a 512-dimensional vector using an embedding algorithm. This embedding step occurs only in the first encoder. Subsequent encoders receive a list of 512-dimensional vectors, where the input to the first encoder consists of word embeddings, and the outputs of each encoder are passed as inputs to the next encoder [26]. This provides a general overview of the model; let's now explore each component in greater detail to understand how it works.

8.2 Attention

In the paper [28], the authors employed the "scaled Dot-Product Attention" mechanism, as illustrated in Fig. 7. The first step in this method is to generate three vectors—Query, Key, and Value—from each input vector in the encoder. This is achieved by multiplying the original embeddings by three distinct matrices, which are initially random and later refined through training. These vectors have a smaller dimensionality than the original embeddings, typically set to 64 (by design), while the embeddings, along with the input and output vectors of the encoder, have a dimensionality of 512.

By multiplying the input vector x_1 with the weight matrix W_Q, we obtain the query vector q_1. Similarly, the key k_1 and value v_1 vectors are generated by multiplying x_1 with their corresponding weight matrices, W_K and W_V.

The next step in the self-attention mechanism is to compute scores for each word in the input sequence. For instance, when processing the first word, "Bird," in the sentence "Bird flew over the lake," we calculate scores for each word relative to "Bird." These scores determine how much attention should be paid to the other words when encoding "Bird."

To calculate these scores, we take the dot product of the query vector for "Bird" with the key vectors of all words. Specifically, for the word in position #1, the first score is the dot product of the query vector q_1 with the key vector k_1. The second score is the dot product of q_1 and k_2, and this continues for each word in the sentence. The score calculation can be expressed as:

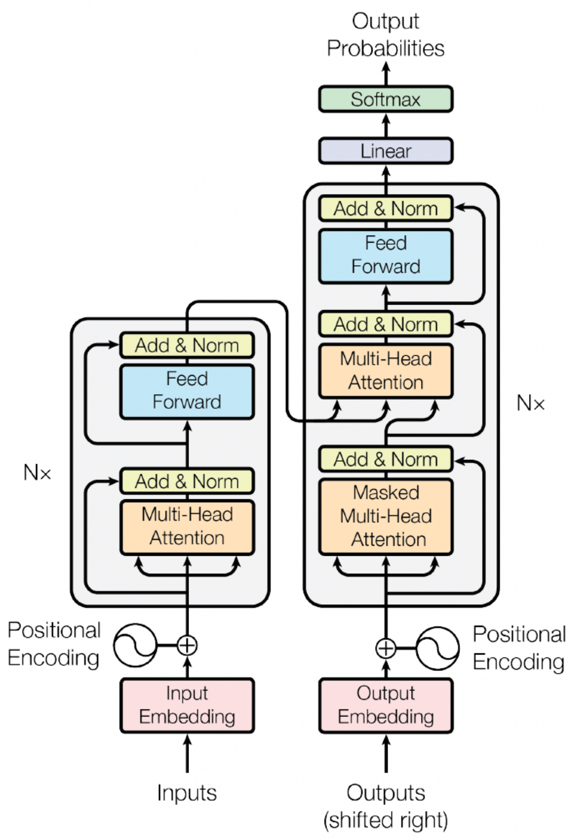

Figure 6: Transformer architecture. Source [28].

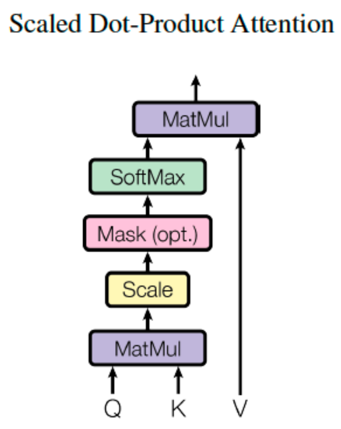

Figure 7: Scaled Dot-Product Attention. Source [28].

$$\text{Score}(i, j) = q_i . k_j \tag{22}$$

Where q_i represents the query vector for the i-th word, and k_j is the key vector for the j-th word.

To stabilize the gradient during training, we normalize the scores by dividing each score by the square root of the dimension of the key vectors. The queries and keys have a dimension of d_k, while the values have a dimension of d_v. In the case described by [28], d_k is 8 (the square root of 64). Finally, we apply the softmax function to the scaled scores to normalize them, converting the scores into probabilities that sum to 1.

Rather than using a single vector for each word, we now consider a matrix of input vectors. For each input, the model generates matrices for the queries, keys, and values. The scaled dot-product attention can therefore be defined as follows:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \tag{23}$$

In this context, Q denotes the matrix of query vectors, K represents the matrix of key vectors, and V is the matrix of value vectors. The paper also introduces the concept of multi-head attention, which enhances the self-attention mechanism by performing multiple attention operations in parallel.

Each attention output is calculated independently, then concatenated and linearly transformed to the desired dimension. This process involves creating multiple sets of query, key, and value vectors, represented as Q^h, K^h, and V^h for each head h. The attention output for each head is computed as follows:

$$\text{Output}^h = \text{Softmax}\left(\frac{Q^h \cdot (K^h)^T}{\sqrt{d_k}}\right) \cdot V^h \tag{24}$$

The output concatenation is carried out using a linear layer to ensure the resulting vector has the required dimensionality. The final output can be expressed as follows:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{Output}^1, \text{Output}^2, \ldots, \text{Output}^H) \cdot W^O \tag{25}$$

where W^O is a learned weight matrix.

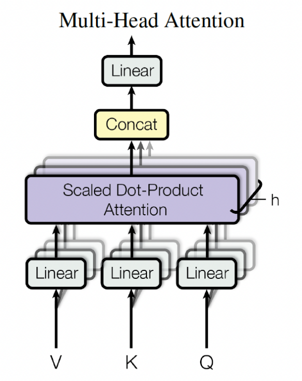

Figure 8: Multi-Head Attention consisting of several attention layers running in parallel. Source [28].

As mentioned earlier, the decoder incorporates an additional encoder-decoder attention layer, which is represented in the masked multi-head attention mechanism. Before applying the softmax function to the product of the query and key matrices, we assign a large negative value to the scores corresponding to future positions (blocking information from tokens to the right of the current position). This ensures that future words do not affect the representation of the current word. This process can be mathematically expressed as follows:

$$\text{Masked Score}(i, j) = \begin{cases} q_i \cdot k_j & \text{if } j \leq i \\ -\infty & \text{if } j > i \end{cases} \tag{26}$$

8.3 Positional encoding

An important feature of this architecture is positional encoding. Since the model does not rely on recurrence or convolution, it requires a method to encode information about the relative or absolute positions of tokens in the sequence, which is crucial for understanding context. Transformers address this need through positional encoding.

Positional encoding involves adding a fixed-size vector to the input embeddings (word embeddings) to represent the relative positions of tokens in the sequence. These position vectors are combined with the token embeddings before being input into the Transformer layers. The encoding is constructed using sine and cosine functions at different frequencies, allowing the model to capture both the sequential order and contextual information effectively [27].

9 Generative Pre-trained Transformers (GPTs)

Traditional deep learning models typically require large amounts of manually labeled data, making them heavily reliant on supervised learning. This dependence poses challenges for tasks with limited labeled data. The Generative Pre-trained Transformer (GPT) was introduced to overcome this limitation by leveraging unsupervised learning [29][31]. Its architecture is designed to utilize massive amounts of unlabeled text data, learning language structures, patterns, and nuances through pre-training, thereby reducing the reliance on labeled datasets for downstream tasks [31].

Like many foundational models, GPT uses transfer learning, which involves two main training stages. First, it undergoes unsupervised pre-training on a vast corpus of text to learn general language patterns, such as grammar, facts, and reasoning. Afterward, it is fine-tuned using supervised learning for specific tasks like question answering or classification. This two-stage process minimizes the need for large labeled datasets, as the pre-training phase equips the model with fundamental language knowledge, which can then be adapted to specific tasks with smaller labeled datasets during fine-tuning.

9.1 Unsupervised pre-training

During pre-training, the model is fed vast amounts of unlabeled text data and learns by predicting the next word in a sequence based only on preceding context, a process known as language modeling. This task is often called self-supervised since sentences serve as both inputs and targets. Here, the input is a text sequence, and the output is a probability distribution over possible next words.

GPT is based on the Transformer Decoder architecture [28], specifically a unidirectional Transformer with masked self-attention and feed-forward layers, designed to generate text in a left-to-right manner. During training, the model processes entire sequences at once, using masked self-attention to ensure each token only attends to previous tokens. However, during inference, the model processes one token at a time to predict the next token in the sequence, adding each generated token back into the input. This approach is known as autoregression [30].

After passing the input sequence through all layers, the model produces a vector representing the predicted next token's probability distribution over the vocabulary. The token with the highest probability is then selected as the output. Theoretically, given an input corpus of tokens $\mathcal{U} = u_1, u_2, ..., u_n$, the objective during training is to maximize the log-likelihood of predicting each token in the sequence given all previous tokens. This is formulated as:

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta) \tag{27}$$

where k is the context window, P is the conditional probability and the parameters of neural network model θ are trained using stochastic gradient descent [32]. GPT's Transformer Decoder of GPT uses its self-attention layers to focus on previous words and generate probable next words.

9.2 Supervised fine-tuning

Fine tuning the pre-trained model on the subtasks is done by adding a task specific layer after the last layer of the pre-trained model. In this step, we apply the parameters obtained from the previous phase to the supervised task. Our labeled dataset $\mathcal{C}$ comprises a sequence of input tokens, $x^1, x^2, \ldots, x^m$, along with a corresponding label y. The inputs are processed through a pre-trained model to derive the activation h_l^m from the final transformer block. This activation is then passed into an additional linear output layer with parameters W_y to predict y:

$$P(y|x^1, ..., x^m) = \text{softmax}(h_l^m W_y). \tag{28}$$

Similar to pre-training, the objective is to maximize the likelihood:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, ..., x^m) \tag{29}$$

By including the language modeling as an auxiliary objective to the fine-tuning stage, the generalization of the supervised model gets improved and convergence accelerates. The final objective that combines the two objective functions L_1 and L_2 with parameter λ is written as:

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C}) \tag{30}$$

There have been new additions to the family of GPT models (GPT-2, GPT-3, GPT-4, etc.) over the years. Although they all retain the same Transformer decoder structure, they differ in several ways. In general, a greater number of layers improves performance. For instance, GPT-2 has 1.5 billion parameters, while GPT-3 scales up to 175 billion. The larger the model, the better it can handle complex linguistic nuances and relationships. Additionally, newer GPT models are trained on increasingly diverse datasets (40 GB for GPT-2 and 570 GB for GPT-3). However, as model size increases, performance can slow down, even though it enables handling of more complex tasks [33].

Another important concept is that some models, such as GPT-3, utilize few-shot, one-shot, and zero-shot learning instead of traditional fine-tuning. While fine-tuning is still possible, GPT-3 is primarily known for its ability to perform tasks by analyzing a small number of examples provided directly in the input. This allows users to prompt GPT-3 with task instructions and examples, eliminating the need for fine-tuning on task-specific datasets [33].

10 Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers, or BERT, introduced by Devlin et al., represents a significant breakthrough in natural language processing (NLP) [33]. Unlike GPT, which processes text in a unidirectional manner, BERT is pretrained to understand the context of a word by considering both its left and right surroundings, utilizing a bidirectional Transformer architecture with unlabeled text. The model is then fine-tuned by adding a single output layer on top of the pretrained model. This bidirectional approach enables BERT to gain a deeper understanding of words based on their full context, making it particularly effective in tasks that require nuanced comprehension, such as question answering, classification, and named entity recognition [33].

BERT's architecture is an encoder-only Transformer model that processes text in parallel across layers, utilizing an attention mechanism that allows it to read the entire sequence of words simultaneously. As a result, it is considered bidirectional, providing a richer context. A common challenge in training language models is next-word prediction, as many models make predictions sequentially, which is inherently directional and limits the amount of context they can learn [35]. To address this, BERT uses two unsupervised tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).

10.1 Masked Language Modeling (MLM)

In this task, a portion of the input tokens is randomly masked, and the model is tasked with predicting these masked tokens. The final hidden vectors corresponding to the masked tokens are passed through a softmax function to calculate the probabilities for the possible candidate words. Since BERT adopts a transfer learning approach that involves pretraining and fine-tuning, a mismatch arises between these stages, as the masked tokens are not present during fine-tuning. To mitigate this, the pretraining data generator randomly selects 15% of token positions for masking [34].

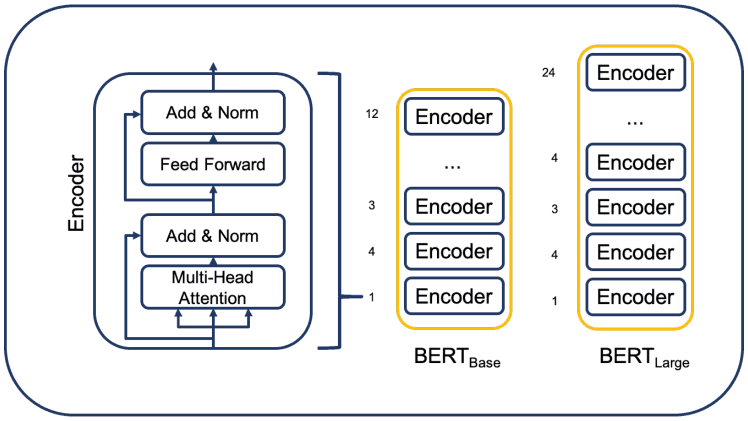

Figure 9: BERT model: a stack of encoder layers. Source unknown.

10.2 Next Sentence Prediction (NSP)

The second unsupervised task (or more precisely, self-supervised, since the output data is also the input data) focuses on understanding the relationship between pairs of sentences, which is useful for tasks such as sentence entailment or information retrieval. During training, half of the input pairs are matching pairs, where the second sentence logically follows the first, while the other half are mismatched, with the sentences randomly paired. To help the model distinguish between the two scenarios, the special [CLS] token is added at the beginning of each input, and the [SEP] token serves as a separator between the two sentences [34].

11 Text-to-Text Transfer Transformer (T5)

The T5 model processes input as a combination of the task description and the corresponding input data, with the output representing the desired target. By unifying all tasks into a "text-to-text" format, T5 learns to map input text to output text across a wide variety of NLP tasks.

This text-to-text approach offers several advantages. First, it simplifies both the training and deployment processes by allowing a single model to handle multiple tasks. This eliminates the need for specialized models for each task, reducing the complexity of developing and maintaining different architectures for various tasks.

In addition, the text-to-text format promotes transfer learning. By training on a variety of tasks, T5 learns generalized language representations and acquires extensive linguistic knowledge. This allows the model to perform effectively on new tasks with limited training data, as it can leverage its understanding of various language patterns and structures.

Moreover, the approach facilitates the use of prompt engineering. Task-specific prompts serve as instructions that guide the model to produce contextually relevant outputs. These prompts help the model adapt to specific tasks during fine-tuning, enhancing its ability to generate accurate results.

Recent advancements in language modeling have explored different architectural variations, including encoder-only models like BERT and decoder-only models seen in many large language models. However, T5 utilizes an encoder-decoder architecture, closely resembling the original Transformer model, which combines the strengths of both components for superior performance across a range of tasks. For a more in-depth explanation of the architecture's design, readers are referred to the original paper [28].

12 Vision Transformers (ViT)

Transformers, originally designed for NLP, have been adapted for computer vision tasks through the Vision Transformer (ViT) model, marking a significant innovation in applying self-attention mechanisms to visual data. Traditional computer vision models, such as Convolutional Neural

Networks (CNNs), rely on spatial hierarchies and convolutional operations to extract features from images. In contrast, ViT replaces convolutions with a self-attention mechanism, treating an image as a sequence of flattened patches.

In ViT, an image is divided into fixed-size patches, which are then linearly embedded into a sequence of vectors, similar to tokenized inputs in NLP models. These patch embeddings are processed through a Transformer architecture, allowing the model to capture global relationships between patches—something that convolutional models struggle with. ViT has demonstrated that self-attention is effective in modeling long-range dependencies in images, achieving competitive performance with CNNs, particularly when trained on large datasets. The success of ViT highlights the adaptability of the Transformer architecture across various domains, from natural language processing to computer vision.

13 Conclusion

In this paper, we provided a comprehensive overview of some of the most prominent models used in natural language processing (NLP) applications. We examined their strengths and limitations, highlighting key applications as well as the challenges they face. Our aim was to offer a concise and accessible resource for those outside the field, serving as an introductory guide to foundational models in NLP. We hope this work will serve as a valuable starting point for individuals seeking to understand and engage with the core practices of modern NLP.

References

[1] Wikipedia contributors. "Georgetown–IBM experiment." *Wikipedia, The Free Encyclopedia.* Available at: `https://en.wikipedia.org/wiki/GeorgetownâĂŞIBM_experiment`.

[2] History of Information. "Markov Chains." *HistoryofInformation.com.* Available at: `https://www.historyofinformation.com/detail.php?id=4137`. Accessed 10 Dec. 2024.

[3] Manning, Christopher D., and Hinrich Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, 1999.

[4] Manning, Christopher D., and Hinrich Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, 1999.

[5] Baum, Leonard E., Ted Petrie, George Soules, and Norman Weiss. "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains." *The Annals of Mathematical Statistics*, vol. 41, no. 1, 1970, pp. 164–171.

[6] Hopfield, John J. "Hopfield Network." *Scholarpedia.* Available at: `http://scholarpedia.org/article/Hopfield_network`. Accessed 10 Dec. 2024.

[7] Chung, Junyoung, et al. "Gated Feedback Recurrent Neural Networks." In: *Proceedings of the 32nd International Conference on Machine Learning - Volume 37*, ICML'15, Lille, France, 2015, pp. 2067–2075.

[8] Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural Turing Machines." 2014.

[9] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long Short-Term Memory." *Neural Computation*, vol. 9, no. 8, 1997, pp. 1735–80.

[10] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long Short-Term Memory." *Neural Computation* 9, no. 8 (December 1997): 1735–1780.

[11] Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation." In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal, 2015, pp. 1412–1421.

[12] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation." In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.* Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1412–1421.

[13] Nicholson, Chris. "A Beginner's Guide to LSTMs and Recurrent Neural Networks." Available at: `https://skymind.ai/wiki/lstm`. Accessed: 06 November 2019.

[14] Schmidt, Robin M. "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview."

[15] Chen, Gang. "A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation."

[16] Zhang, Aston, et al. *Dive into Deep Learning.* Available at: http://www.d2l.ai. 2019.

[17] "Backpropagation Through Time Explained with Derivations." *Pycodemates*, 2023. Available at: https://www.pycodemates.com/2023/08/backpropagation-through-time-explained-with-derivations.html.

[18] "Understanding LSTMs." *Colah's Blog*, 2015. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[19] Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional Recurrent Neural Networks." *IEEE Transactions on Signal Processing*, vol. 45, 1997, pp. 2673–2681.

[20] Kazemnejad, Reza. "Transformer Architecture and Positional Encoding." Available at: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.

[21] Sutskever, Ilya, Oriol Vinyals, and Quoc V Le. "Sequence to Sequence Learning with Neural Networks." In: *Advances in Neural Information Processing Systems 27.* Curran Associates, Inc., 2014, pp. 3104–3112.

[22] Chiu, Chung-Cheng. "State-of-the-Art Speech Recognition with Sequence-to-Sequence Models."

[23] Venugopalan, Subhashini, et al. "Sequence to Sequence – Video to Text." Available at: https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/.

[24] Weng, Lilian. "Attention Mechanisms." Available at: https://lilianweng.github.io/posts/2018-06-24-attention/.

[25] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In: *3rd International Conference on Learning Representations, ICLR 2015*, San Diego, CA, USA, May 7-9, 2015.

[26] "Transformers: An Overview." *Towards Data Science*. Available at: https://towardsdatascience.com/transformers-141e32e69591.

[27] Kazemnejad, Reza. "Transformer Architecture and Positional Encoding." Available at: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.

[28] Vaswani, Ashish, et al. "Attention Is All You Need." In: *Proceedings of NeurIPS 2017.*

[29] Tsvetkov, Yulia. "Opportunities and Challenges in Working with Low-Resource Languages." CMU, 2017.

[30] "Illustrated GPT-2." *Jay Alammar's Blog*. Available at: https://jalammar.github.io/illustrated-gpt2/.

[31] Radford, Alec, et al. "Improving Language Understanding by Generative Pre-Training."

[32] Robbins, H., and S. Monro. "A Stochastic Approximation Method." *The Annals of Mathematical Statistics*, 1951, pp. 400–407.

[33] Huiming, Song. "GPT-1, GPT-2, GPT-3, InstructGPT, ChatGPT, and GPT-4: A Summary." Available at: https://songhuiming.github.io/pages/2023/05/28/gpt-1-gpt-2-gpt-3-instructgpt-chatgpt-and-gpt-4-summary/.

[34] Devlin, Jacob, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding."

[35] "BERT Explained: State-of-the-Art Language Model for NLP." *Towards Data Science*. Available at: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[36] i2Tutorials. "What is the Difference Between Bidirectional RNN and RNN?" i2tutorials.com. Available at: https://www.i2tutorials.com/what-is-the-difference-between-bidirectional-rnn-and-rnn/. Accessed 10 Dec. 2024.

[37] Aman AI. "Deep Learning Components Primer." Aman.ai. Available at: https://aman.ai/primers/ai/dl-comp/. Accessed 10 Dec. 2024.

[38] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks." arXiv, 2014, https://arxiv.org/abs/1409.3215. Accessed 10 Dec. 2024.

[39] Jalammar, Jay. "The Illustrated Transformer." The Illustrated Transformer, 2018, https://jalammar.github.io/illustrated-transformer/. Accessed 10 Dec. 2024.