International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT217696

Scope and Challenges in Conversational AI using Transformer Models

Arighna Chakraborty, Asoke Nath
Department of Computer Science, St. Xavier's College (Autonomous) Kolkata, India

Article Info
Volume 7, Issue 6
Page Number : 372-384

Publication Issue :
November-December-2021

Article History
Accepted : 12 Dec 2021
Published : 26 Dec 2021

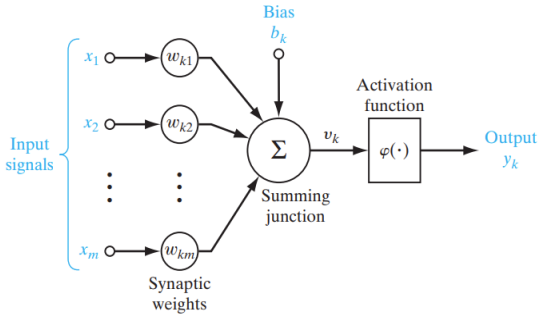
ABSTRACT

Conversational AI is an interesting problem in the field of Natural Language Processing and combines natural language processing with machine learning. There has been quite a lot of advancements in this field with each new model architecture capable of processing more data, better optimisation and execution, handling more parameters and having higher accuracy and efficiency. This paper discusses various trends and advancements in the field of natural language processing and conversational AI like RNNs and RNN based architectures such as LSTMs, Sequence to Sequence models, and finally, the Transformer networks, the latest in NLP and conversational AI. The authors have given a comparison between the various models discussed in terms of efficiency/accuracy and also discussed the scope and challenges in Transformer models.

Keywords – deep learning, neural networks, recurrent neural networks, long short term memory, sequence to sequence, transformer models, switch transformer models

I. INTRODUCTION

Neural networks are famous for mimicking the way a human mind functions or in a more elaborate sense, the way biological neurons signal one another. Neural networks, also knows as artificial neural networks (ANNs) comprises of layers of nodes, generally consisting of an input layer, numerous hidden layers and an output layer [2].



$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

$$y_k = \varphi(u_k + b_k)$$

Structure of a simple Neural Network[3]

Once an input layer is determined, weights are assigned. These weights help determine the importance of any given variable, with larger ones contributing more significantly to the output compared to other inputs [3]. All inputs are then multiplied by their respective weights and then summed. Afterward, the output is passed through an activation function, which determines the output. If that output exceeds a given threshold, it "fires" (or activates) the node, passing data to the next layer in the network. This results in the output of one node becoming in the input of the next node.

The manner in which the nodes in a neural network are structured is intimately linked with the learning algorithm used to train the network. In general, there are a few different classes of network architectures:

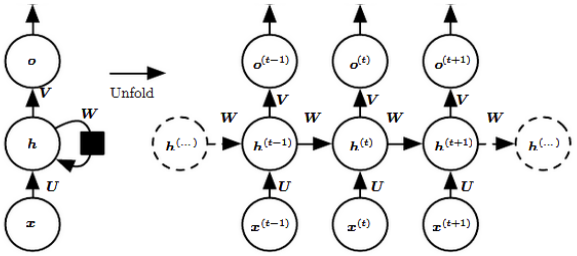
- Single-layer Feedforward Networks
- Multi-layer Feedforward Networks
- Recurrent Neural Networks

Conversational AI models explicitely make use of recurrent neural networks (RNNs) or a modification of the network architecture. In traditional neural networks it is generally assumed that all inputs (and outputs) are independent of each other, but for many tasks that is a bad idea. In order to predict the next word in a sentence, it is essential to know which words came before it. [4]

II. Recurrent Neural Networks (RNN)

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. They are incorporated into popular applications such as Siri, voice search, and Google Translate [5]. They are distinguished by their "memory" as they take information from prior inputs to influence the current input and output. RNNs are called recurrent because they perform the same task for every element in a sequence, with the output being dependent on the previous computations and inputs.

Another distinguishing characteristic of recurrent networks is that they share parameters across each layer of the network. While feedforward networks have different weights across each node, recurrent neural networks share the same weight parameter within each layer of the network.



Recurrent Neural Network (RNNs)[6]

- **Input:** x(t) is taken as the input to the network at time step t. For example, x1, could be a vector corresponding to a word of a sentence.
- **Hidden state**: h(t) represents a hidden state at time t and acts as "memory" of the network. h(t) is calculated based on the current input and the previous time step's hidden state:

$$h(t) = f(U.x(t) + W.h(t-1))$$

The function f is taken to be a non-linear transformation such as tanh, ReLU.

- **Weights**: The RNN has input to hidden connections parameterized by a weight matrix U, hidden-to-hidden recurrent connections parameterized by a weight matrix W, and hidden-to-output connections parameterized by a weight matrix V and all these weights (U,V,W) are shared across time.

- **Output**: o(t) illustrates the output of the network. In the figure there are further arrows

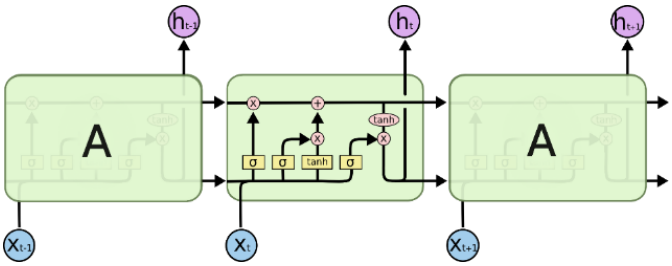
after o(t) which is also often subjected to non-linearity,especially when the network contains further layers downstream **[6]**.

Recurrent neural network utilises Backpropagation Through Time algorithm (BPTT) to determine the gradients (slope of the loss function), which is slightly different from traditional backpropagation. The main principle of BPTT is similar to traditional backpropagation, where the model trains itself by calculating errors from its output layer to its input layer **[5]**. BPTT differs from the traditional approach in that BPTT sums errors at each time step whereas feedforward networks do not need to sum errors as they do not share parameters across each layer.

III. Long Short Term Memory Networks (LSTMs)

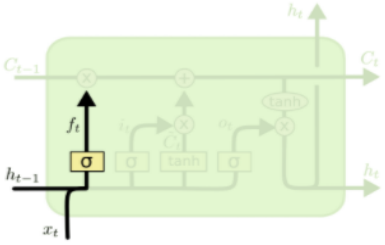
Long Short Term Memory networks – usually called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. The issue of gradients approaching zero due to BPTT is aptly termed as the '**vanishing gradients problem**', and it was for many years the chief issue with recurrent neural networks, limiting their ability to learn correlations between inputs that were temporally distant from one another (particularly long sequences of data) **[9].**

The key to LSTMs is 'cell state' – the horizontal line at the top. For the most part, the purpose of the cell state is to pass information down the chain without transforming it. The LSTM does have the ability to remove or add information to the cell state from the previous states, carefully regulated by structures called gates. Gates optionally allow information through, and are made of sigmoid activated neural network layer and a pointwise multiplication operation **[9]**.An LSTM has protect and control the cell state

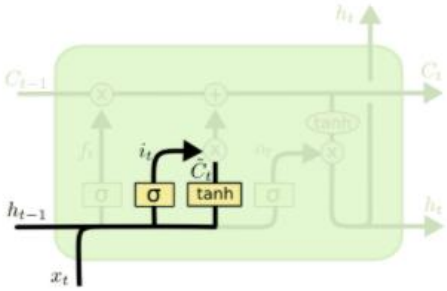


Understanding LSTM Networks -- colah's blog[9]

The first step is to decide what information is not relevant to be considered in the next hidden layers and will be "thrown away" from the cell state. This decision is made by the first gate layer consisting of a sigmoid layer called the "forget gate layer". For each number in the cell state vector, the forget gate outputs a number between zero and one.

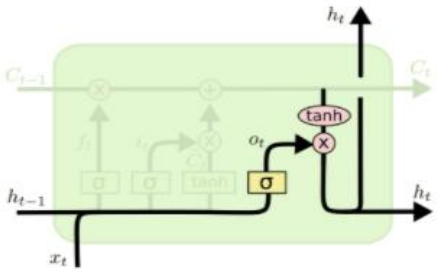


The second gate layer is responsible for adding features from the current input to the cell state. The second layer further contains two parts – first the "input gate layer" which is another sigmoid layer that produces values between zero and one, determining which values will be updated. The second part of the layer employs a tanh activation function to produce new values to be added to cell state. After values are produced by the sigmoid and tanh, they are point-wise multiplied



Finally, the output of the cell state is generated which will be forwarded to the next layer. Unsurprisingly, a sigmoid layer is used to output values between 0 and

1, determining which components of cell state will be part of the next hidden state (based on relevance to the input at hand) [9].

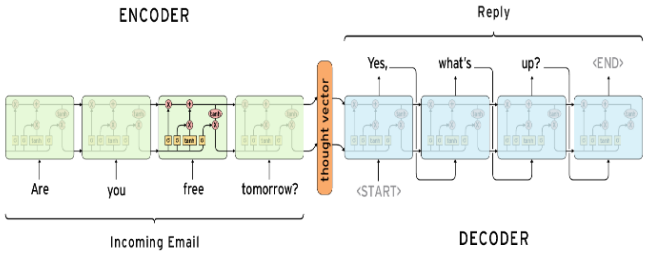


This decoupling of cell state and hidden state is noteworthy, because it means that the network can remember features in cell state for longer periods of time without including them in the hidden state that affects the current prediction [8]

IV. Sequence to Sequence Models (Seq2Seq)

Despite their flexibility and power, deep neural networks can only be applied to problems whose inputs and targets can be sensibly encoded with vectors of fixed dimensionality. It is a significant limitation since many important problems are best expressed with sequences whose lengths are not known a-priori [12]. For example, speech recognition and machine translation are sequential problems. Likewise, question answering can also be seen as mapping a sequence of words representing the question to a sequence of words representing the answer. It is therefore clear that a domain-independent method that learns to map sequences to sequences would be useful.

Sequence to sequence models are a straightforward application of the Long Short Term Memory architecture. A novel neural network architecture that learns to encode a variable-length sequence into a fixed-length vector representation and to decode a given fixed-length vector representation back into a variable-length sequence [13].



Chatbots with Seq2Seq (suriyadeepan.github.io)[14]

The idea is to use one LSTM (called the encoder) to read the input sequence, one time step at a time, to obtain fixed dimensional vector representation, and then use another LSTM (called the decoder) to extract the output sequence from that vector. Each hidden state influences the next hidden state and the final hidden state is called the context or thought vector since it represents the intention of the sequence. From the context vector, the decoder generates another sequence, one symbol at a time.

In order to solve the variable length problem, the concept of padding was introduced. Prior to training, the dataset is modified from variable length sequences to fixed length sequences. Special symbols are used to fill in the sequences. For example,

EOS: End of sentence,
PAD: Filler,
UNK: Unknown Symbol.

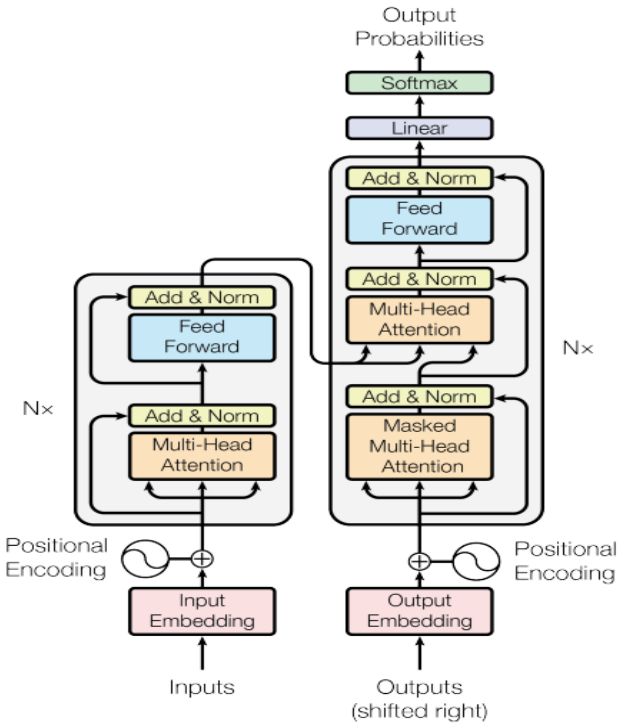
Introduction of padding did solve the problem of variable length sequences but for datasets with very large sequences, the smaller sequences will require a lot of pad symbols in the encoded version. This might overshadow the actual information in the sentence. Bucketing aims to solve this problem, by putting the sequences into buckets of different sizes [14].

V. Transformer Models

Most sequence generation models have an encoder-decoder structure (as in sequence to sequence model). In the case of the Transformer model, the encoder

maps an input sequence of symbol representations (x1, . . . , xn) to a sequence of continuous representations z = (z1, ..., zn). Given z, the decoder then generates an output sequence (y1, . . . , ym) of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next[15].

The Encoder and Decoder models used in the Transformer do not use LSTM, GRU or RNNs hence there are no recurrent connections and thus no "memory" of previous states are implemented. Transformers get around this lack of memory by perceiving entire sequences simultaneously.



Attention Is All You Need - Transformer Model[15]

V.1 Input Embeddings

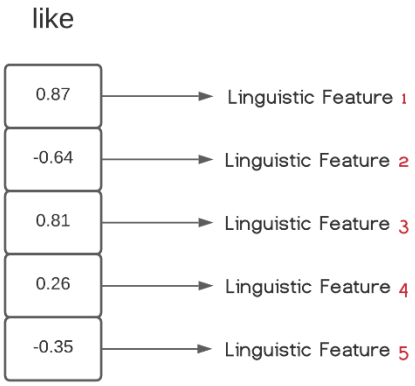
Here is a detailed visualisation of the embedding process in Natural Language Processing models. Consider an input from the user,

Input – Do you like Game of Thrones

The first step is to fetch the indices of the words occurring in the input sequence from the vocabulary of all the learned words by the model. Therefore we get,

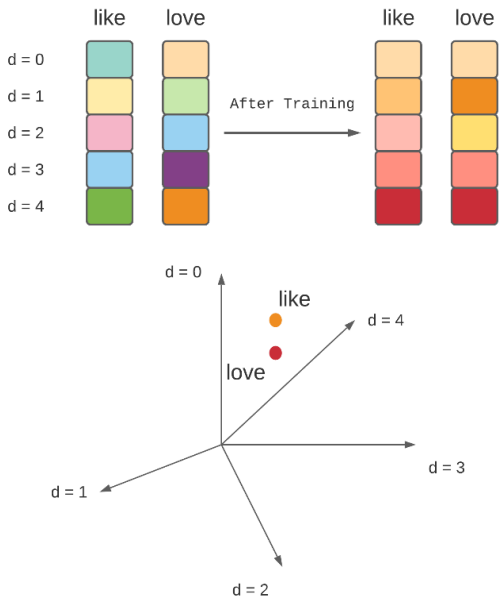
Vocabulary Indices – [2456, 56, 6674, 5345, 86, 145]

Now these indices are fed as the input to the Input Embedding Module of the network. Here, against each of these word indices obtained from the vocabulary, a vector is generated. Initially, the vectors are filled with random numbers. Later on, during training, the model updates them. These vectors are called word embeddings. In the transformer model presented in the paper, "Attention is All you Need", the embedding size or the length of these vectors are 512.



Example of a word embedding

Each element of the vector or in other words, each "dimension" of the word embedding tries to capture a unique linguistic feature for that word. These could be things like whether it is a verb or a pronoun or something else. Now, n-dimensional (here it's 5-dimensional) word embeddings can be represented on a n-dimensional hyperspace, where words sharing similar linguistic features are plotted closer to each other while dissimilar word embeddings are plotted farther apart from each other.



Word Embeddings and their representation

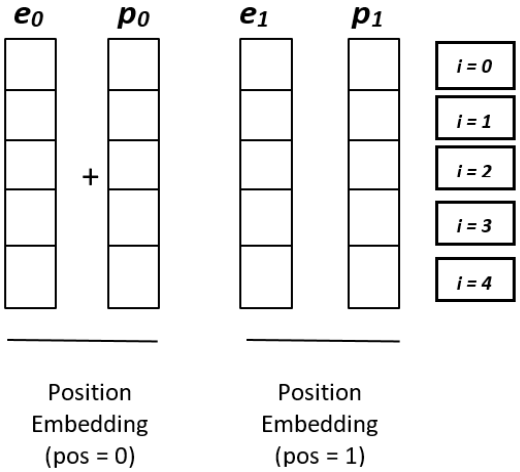
Hence, the main purpose of the embedding layer is to select the proper embedding of the input words and pass them on to the positional encoding module.

V.2 Positional Encoding

Position and order of words in a sentence is an essential part of any language. They define the grammar and thus the actual semantics of a sentence. Recurrent Neural Networks (RNNs) inherently take the order of word into account i.e. they parse a sentence word by word in a sequential manner, but the Transformer architecture ditched the recurrence mechanism in favor of multi-head self-attention mechanism (discussed later). Avoiding the RNNs' method of recurrence will result in massive speed-up in the training time and theoretically, it can capture longer dependencies in a sentence[20].

As each word in a sentence simultaneously flows through the Transformer's encoder or decoder stack (layers of encoder or decoder), the model by itself does not have any sense of the position or order for each word. Consequently, there's still the need for a way to add some information about the positions into the input embeddings.

So a new set of vectors called the position embeddings are introduced, one for each word embedding. Thay have the same embedding size (vector length) as that of the word embeddings. Generally, the position embeddings and the word embeddings are simply added to generate a new set of embeddings; only this time these embeddings also contain the position as well as the linguistic information of the words in the sentence.



Positional Embeddings

The tricky part here is to set the values of the various position embeddings. The authors of the paper "Attention is All you Need" came up with a clever trick to use wave functions (sine and cosine) to capture position information. Here,

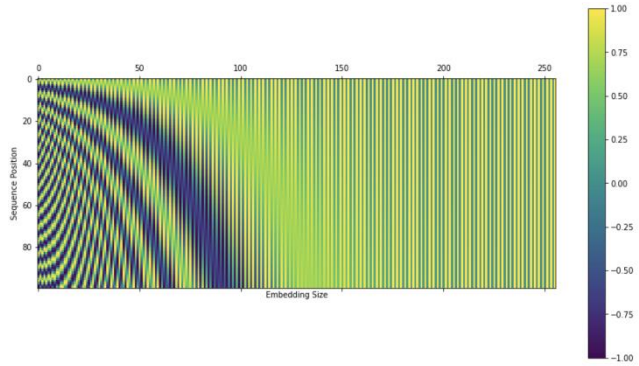
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

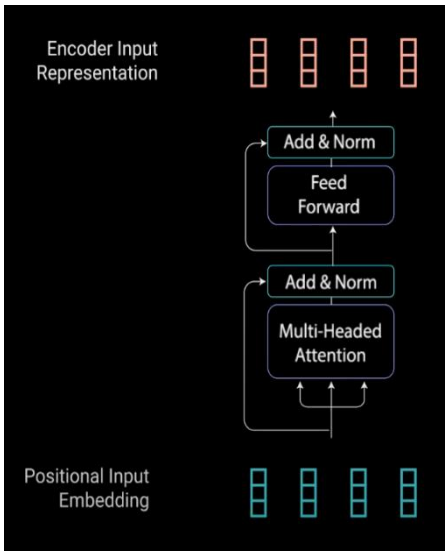
- pos is the position index of the word in the input sequence.
- dmodel is the embedding size (which is 512).
- i represents the indices of each of the position embedding dimensions

So, basically, for every odd value of pos on the position vector, create a vector using the cosine

function. For every even value of pos, create a vector using the sine function



Visualization of Positional Embeddings when plotted using sine and cosine functions [20]



Encoder Layer[20]

Here is how the position embedding curves looks like when plotted in full scale against the embedding size (i). The above plot is a great way to visualise the position embedding dimensions (or values) for a particular embedding size. Finally, the position embeddings are added to their corresponding word embeddings.

From now on, the final embeddings of the words will be referred to as the position embeddings for easier understanding.

V.3 Encoder Stack

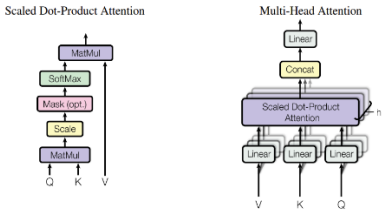
This module consists of a stack of N Encoder Layers. These encoder layers, each of which has two sublayers – the Multi-Head Attention Layer and the Position-wise Full Connected Feed-Forward Network layer. Finally each encoder layer has an Add-Norm (Addition and Normalization) layer component.

All the operations inside each encoder stack are implemented to encode the input embeddings to a continuous representation with attention information. This will help the decoder focus on the appropriate words in the input during the decoding process. The encoder can be stacked on top of each other N times to further encode the information, where each stack has the opportunity to learn different attention representations therefore potentially boosting the predictive power of the transformer network and offer more parallelization.

V.3.1 Multi Head Attention Layer

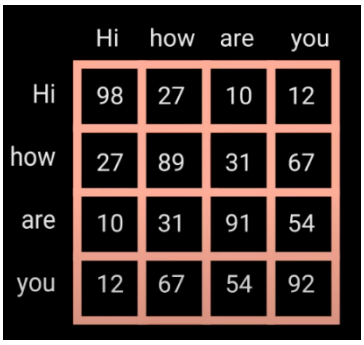
The Multi-Head Attention layer in the encoder applies a specific attention mechanism called the self-attention. The task of self attention is to find out the relationship of each word in a sequence with all the other words in that sequence.

The input to the attention layer, in the case of encoders, is the position embedding matrix. Three identical copies of the input embeddings are made – Query (Q), Key (K) and Value (V). The concept of query, key and value comes from retrieval systems like search engines.



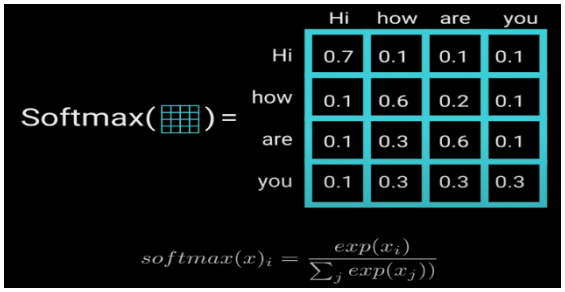
Scaled Dot Product Attention[18]

After feeding the query, key and value matrices through a linear layer, the queries and keys undergo a dot-product matrix multiplication operation to produce a score matrix. The score matrix determines how much focus should a particular word put on other words. So each word will have a score that corresponds to other words in the time-step. The higher the score the more focus. This is how the queries are mapped to the keys.



Example of a Score Matrix[20]

Then these scores are scaled down by dividing each score by the square root of the dimension of the query and key (which is dk). The scaling is done in order to achieve more stable and normalized gradients since multiplication can have exploding effects.



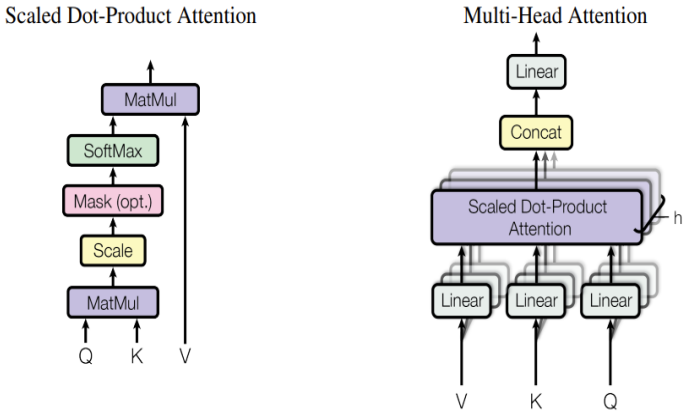
Softmax operation on the scaled scores to get attention weights[20]

A softmax of the scaled scores is performed to get the attention weights, which outputs probability values between 0 and 1. By doing a softmax the higher scores get elevated, and lower scores are depressed. This allows the model to be more confident about which words to attend too.

Finally, the attention weights are multiplied to the value matrix to get an output matrix. The higher softmax scores will keep the value of words the model learns is more important. The lower scores will drown out the irrelevant words. The following equation summarizes the whole process,

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

The query, key and value matrices go through the self-attention process individually. Each self-attention process is called a head. Each head produces an output vector that gets concatenated into a single vector before going through the final linear layer. In theory, each head would learn something different therefore giving the encoder model more representation power.



Multi-Head Attention Layer[18]

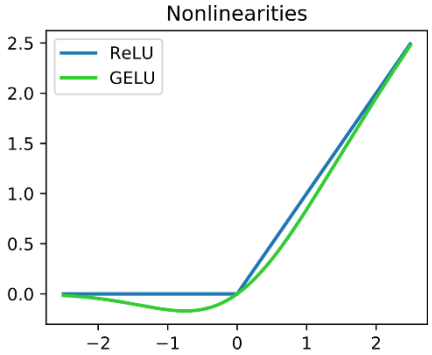
A critical and apparent disadvantage of this fixed-length context vector design as used in sequence to sequence models has the incapability of the system to remember longer sequences. It often forgets the earlier parts of the sequence once it has processed the

entire sequence. The attention mechanism was born to resolve this problem. [18]

The multi-headed attention output vector is added to the original positional input embedding. This is called a residual connection. The output of the residual connection goes through a layer normalization.

V.3.2 Feed Forward Network and Normalization

The normalized residual output gets projected through a pointwise feed-forward network for further processing. The pointwise feed-forward network is a couple of linear layers with a ReLU (Rectified Linear Unit) activation function in between. The output of that is then again added to the input of the Add and Normalize layer and further normalized.



Rectified Linear Unit activation function

The residual connections help the network train, by allowing gradients to flow through the networks directly. The layer normalizations are used to stabilize the network which results in substantially reducing the training time necessary. The pointwise feedforward layer is used to project the attention outputs potentially giving it a richer representation.

V.4 The Decoder Stack

The main purpose of the decoder is to generate text sequences. The decoder stack also comprises multiple decoder layers. Each decoder layer has similar sub-layers as the encoder layers – it has two multi-head attention layers (one is masked, discussed later), a pointwise feed-forward layer, and layer normalization (add and normalization) layers after each sub-layer.

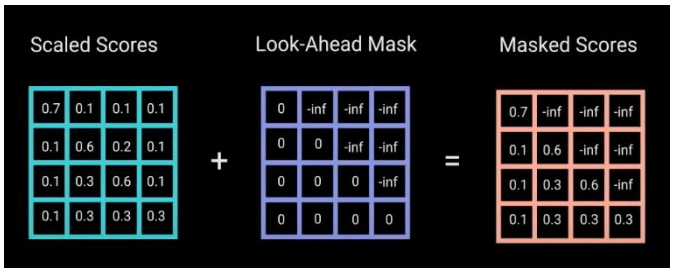
These sub-layers behave similarly to the layers in the encoder but each multi-headed attention layer has a different job. The decoder is autoregressive, it begins with a start token, and it takes in a list of previous outputs as inputs, as well as the encoder outputs that contain the attention information from the input[20].

The beginning of the decoder is pretty much the same as the encoder. The input goes through an embedding layer and positional encoding layer to get positional embeddings. The positional embeddings get fed into the first multi-head attention layer which computes the attention scores for the decoder's input only.

V.4.1 Masked Multi-head Attention Layer

This multi-headed attention layer operates slightly differently. Since the decoder is autoregressive and generates the sequence word by word, the model needs to prevent it from taking future words from the encoder into account[20]. This is true for all other words, where they can only attend to previous words.

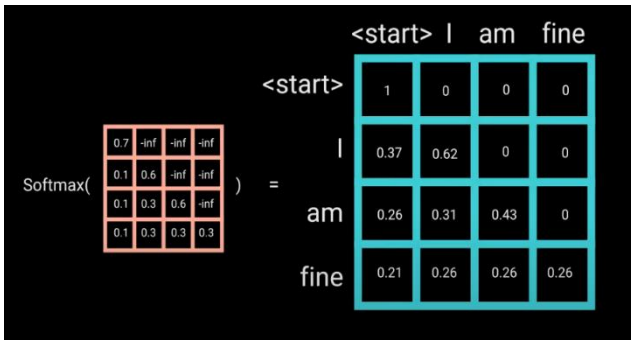
Therefore, the model needs to prevent computing attention scores for future word embeddings. This method is called masking and hence this attention layer is called Masked Multi-Head Attention Layer. To prevent the decoder from taking future words into account, a look-ahead mask is applied to the scaled score matrix. The look-ahead mask is a matrix with the same size as the score matrix with values of zeros and negative infinities only. The mask is simply added to the scales score matrix, and a masked score matrix is obtained.



Using Look-Ahead Mask to generate Masked Score Matrix[20]

The reason for the mask is because once the model calculates the softmax of the masked scores, the negative infinities get zeroed out, leaving zero attention scores for future tokens. This essentially tells the model to put no focus on those words whose attention scores are zero. This masking is the only difference in how the attention scores are calculated in the first multi-headed attention layer.

This layer still has multiple heads that the mask is being applied to, before getting concatenated and fed through a linear layer for further processing. The output of the first multi-headed attention is a masked output vector with information on how the model should attend to the decoder's input.



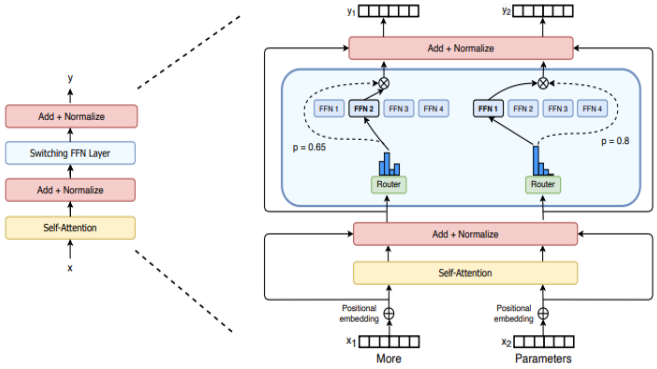
Softmax Operation on the Masked Score Matrix[20]

For the second multi-headed attention layer, the encoder's outputs are the queries and the keys, and the first multi-headed attention layer outputs act as the values. This process matches the encoder's input to the decoder's input, allowing the decoder to decide which encoder input is relevant to put a focus on. The output of the second multi-headed attention goes through a pointwise feedforward layer for further processing.

The Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output allows for significantly more parallelization [15] and can reach a new state of the art in translation quality.

VI. Switch Transformers

Large scale training has been an effective path towards flexible and powerful neural language models. Simple architectures – backed by a generous computational budget, dataset size and parameter count – surpass more complicated algorithms. Inspired by the success of model scale, but seeking greater computational efficiency, a sparsely-activated expert model was proposed: the Switch Transformer [20]



Switch Transformer Encoder Block (arXiv:2101.03961)

The guiding design principle for Switch Transformers is to maximize the parameter count of a Transformer model in a simple and computationally efficient way. Importantly, this work advocates training large models on relatively small amounts of data as the computationally optimal approach.

In a Switch Transformer feed-forward neural network (FFN) layer, each token passes through a router

function which sends it to a single FNN, known as an 'expert.' While each token passes through a single FFN, thecomputation does not increase with the number of experts. "We replace the dense feed-forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer. The layer operates independently on the tokens in the sequence. We diagram two tokens (x1 = "More" and x2 = 'Parameters' in the figure) being routed across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value," wrote Google researchers [20].

VII. Result and Discussion

Transformer based self-supervised pre-trained models have transformed the concept of Transfer learning in Natural language processing (NLP) using Deep learning approach. Self-attention mechanism made transformers more popular in transfer learning across a broad range of NLP tasks [21].

An experiment as carried out and published in the paper by Xiaoyu Yin, Dagmar Gromann and Sebastian Rudolph [23], which used different datasets to train RNNs based models, CNN based models and Transformer models and compare their performance, accuracy and BLEU scores. BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another. Although the models were not trained on large sequences of data, it is still an interesting experiment.

	Mon		Mon80		Mon50		LC-QUAD		DBNQA	
Models	V	T	V	T	V	T	V	T	V	T
NSpM	80.43	80.28	87.55	87.03	85.19	85.54	43.91	43.50	65.89	65.92
NSpM+Att1	80.36	80.58	87.82	87.34	85.98	86.17	52.68	50.13	89.87	89.87
NSpM+Att2	80.88	80.03	87.99	87.37	86.60	86.52	53.03	50.86	91.51	91.50
GNMT-4	80.12	79.53	85.94	85.39	82.92	83.01	43.69	42.71	69.65	69.61
GNMT-8	79.30	79.07	84.94	84.14	80.35	80.76	44.32	43.91	68.43	68.41
LSTM_Luong	92.39	91.67	96.35	96.12	94.05	94.75	52.43	51.06	77.64	77.67
ConvS2S	**98.35**	**97.12**	**96.74**	**96.47**	**96.44**	**96.62**	**61.89**	**59.54**	**96.05**	**96.07**
Transformer	95.25	95.31	95.16	94.87	93.80	93.92	58.99	57.43	68.68	68.82

Table 1 - BLEU scores for all models[22]

	Mon				Mon80				Mon50				LC-QUAD				DBNQA			
Models	V		T		V		T		V		T		V		T		V		T	
NSpM	71	95	75	93	75	95	76	95	82	97	79	96	0	61	0	61	0	77	0	77
NSpM+Att1	71	95	75	93	77	96	78	96	83	97	82	97	1	68	1	66	63	93	63	93
NSpM+Att2	73	96	74	92	79	97	78	96	84	97	81	97	1	68	1	67	69	94	69	94
GNMT-4	70	95	71	92	67	95	68	95	77	96	75	96	0	62	0	61	1	84	1	84
GNMT-8	68	95	73	91	58	94	60	94	74	96	71	95	0	65	0	64	0	84	0	84
LSTM_Luong	75	94	76	94	82	95	84	96	**90**	**98**	89	97	0	68	0	67	34	82	34	82
ConvS2S	**94**	**99**	**95**	**96**	**91**	**98**	**90**	**98**	**89**	**98**	**90**	**98**	**8**	**74**	**8**	**73**	**85**	**98**	**85**	**97**
Transformer	88	98	91	95	83	96	84	96	86	92	84	92	7	71	4	70	3	79	3	80

Table 2 - Accuracy (in %) | F1 score[22]

In the above tables, T represents training scores and V refers to validation scores. The rows represent the scores of different RNN based and ConS2S models and finally, the transformer model. While the columns refer to the different datasets on which these models were trained and validated.

In Table 1, BLEU scores are reported on the validation V and test T set for each dataset for the best performing version of each model. ConvS2S outperforms all other models on most datasets. Similarily in Table 2, there are some cases where Transformer outperforms ConS2S in terms of Accuracy. It is evident that, although Transformer is proved as the best model to handle really long sequences, the RNN and CNN based model could still work very well or even better than Transformer in the short-sequences task.

Another experiment in Machine Translation presented in the paper "Attention is all you Need" [15], the participating models were trained on the standard WMT 2014 English-German dataset consisting of about 4.5 million sentence pairs. For English-French, a significantly larger WMT 2014 English-French dataset consisting of 36M sentences and split tokens into a 32000 word-piece vocabulary was used. Sentence pairs were batched together by approximate sequence length. Each training batch contained a set of sentence pairs containing approximately 25000 source tokens and 25000 target tokens.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

BLEU scores on the English-to-German and English-to-French newstest2014 tests [15]

On the WMT 2014 English-to-German translation task, the big transformer model (Transformer (big)) outperforms the best previously reported models (including ensembles) by more than 2.0 BLEU, establishing a new state-of-the-art BLEU score of 28.4. Even the base model surpasses all previously published models and ensembles, at a fraction of the training cost of any of the competitive models. On the WMT 2014 English-to-French translation task, the big model achieves a BLEU score of 41.0, outperforming all of the previously published single models, at less than 1/4 the training cost of the previous state-of-the-art model.

This is to be noted that the experiments conducted are all based on the base transformer model. Since then, based on the basic architecture, researchers have introduced BERT, RoBERTa, ALBERT, Transformer XL, and many more state-of-the-art models. These models further improve upon the underlying transformer architecture in many aspects providing higher accuracy and better performance.

VIII. Conclusion and Future Scope

The recent developments in language modeling offer a lot of improvements in the field of Natural Language Processing. The transformer architecture has become the preferred deep-learning model for conversational AI research. Many efforts have been towards increasing the size of these models, primarily measured in the number of parameters. For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers [17]. While these models provide excellent results, further efforts can make them adapt better to a specific domain.

BAAI's 1.75 trillion parameters, Wu Dao 2.0 and OpenAI's GPT-3 175 billion parameters, alongside HuggingFace DistilBERT and Google GShard, are other popular language models. Compared to Google's T5 NLP model, the baseline version of the Switch Transformer achieved a target pre-training perplexity metrics in 1/7 the training time. It also outperformed a T5-XXL on the perplexity metric, with comparable or better performance on downstream NLP tasks, despite training on half of the data.

IV. REFERENCES

[1]. IBM Cloud Education,"Conversational AI", 31st August 2020.
[2]. IBM Cloud Education,"Neural Networks", 17th August 2020.
[3]. Simon Haykin, "Neural Networks and Learning Machines", Pearson Education.
[4]. Denny Britz, "Recurrent Neural Network Series", 17th September 2015.
[5]. IBM Cloud, "Recurrent Neural Networks", 14th September 2020.
[6]. Cem Dilmegani,"In-Depth Guide to Recurrent Neural Networks (RNNs) in 2021", 16th November 2021
[7]. Javaid Nabi, "Recurrent Neural Networks (RNNs)", 12th July 2019.
[8]. I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning". MIT Press, 2016.
[9]. Pranav Pillai, "Recurrent Neural Networks, the Vanishing Gradient Problem, and Long Short-Term Memory", 17th July 2019.
[10]. Christopher Olah, "Understanding LSTM Networks", 27th August 2015.

[11]. Cho, K.; Merrienboer, B.; Gülçehre Ç.; Bougares, F.; Schwenk, H.; Bengio, Y., "Learning Phrase Representations using RNN". In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014.

[12]. Marc Moreno Lopez, Jugal Kalita, "Deep Learning applied to NLP", arXiv:1703.03091.

[13]. Ilya Sutskever, Oriol Vinyals, Quoc V.Le, "Sequence to Sequence Learning with Neural Networks",10th September 2014.

[14]. Kyunghyun Cho, Bart vanMerrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", 3rd June 2014.

[15]. Jonas Gehring Michael Auli David Grangier Denis Yarats Yann N. Dauphin,"Convolutional Sequence to Sequence Learning", arXiv:1705.03122, 25th July 2017

[16]. Suriyadeepan Ram, "Chatbots with Seq2Seq", 28th June 2016.

[17]. MathsWorks.com,"Visualize Word Embeddings Using Text Scatter Plots"

[18]. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin, "Attention is All You Need", 6th Dec 2017.

[19]. Maxime, "What is a Transformer ?", Inside Machine Learning, 4th January 2019.

[20]. Michael Phi, Illustrated Guide to Transformers- Step by Step Explanation, 1st May 2020

[21]. Buomsoo Kim, "Attention in Neural Networks", 11th November 2020.

[22]. Harshall Lamba,"Intuitive Understanding of Attention Mechanism in Deep Learning", Towards Data Science, 20th March 2019.

[23]. Ketan Doshi, "Transformers Explained Visually", Towards Data Science, 17th January 2017.

[24]. Saurav Singla , Ramachandra N., "Comparative Analysis of Transformer Based Pre-Trained NLP Models", 30th November 2020.

[25]. Xiaoyu Yin, Dagmar Gromann, Sebastian Rudolph, "Neural Machine Translating from Natural Language to SPARQL", 21st June 2019.

Cite this article as :

Arighna Chakraborty, Asoke Nath, "Scope and Challenges in Conversational AI using Transformer Models", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 7 Issue 6, pp. 372-384, November-December 2021. Available at
doi : https://doi.org/10.32628/CSEIT217696
Journal URL : https://ijsrcseit.com/CSEIT217696

Author's Profile

Dr. Asoke Nath is working as Associate Professor in the Department of Computer Science, St. Xavier's College (Autonomous), Kolkata. He is engaged in research work in the field of Cryptography and Network Security, Steganography, Green Computing, Big data analytics, Li-Fi Technology, Mathematical modelling of Social Area Networks, MOOCs, Quantum Computing etc. He has published more than 257 research articles in different Journals and conference proceedings.

Arighna Chakrborty is the final year student of Masters in Computer Science St. Xaviers College (Autonomous), Kolkata. Currently he is involved in Web Development, App Development, Machine Learning.