On the Computational Power of Transformers and its Implications in Sequence Modeling

Satwik Bhattamishra   Arkil Patel   Navin Goyal
Microsoft Research India
{t-satbh,t-arkpat,navingo}@microsoft.com

Abstract

Transformers are being used extensively across several sequence modeling tasks. Significant research effort has been devoted to experimentally probe the inner workings of Transformers. However, our conceptual and theoretical understanding of their power and inherent limitations is still nascent. In particular, the roles of various components in Transformers such as positional encodings, attention heads, residual connections, and feedforward networks, are not clear. In this paper, we take a step towards answering these questions. We analyze the computational power as captured by Turing-completeness. We first provide an alternate and simpler proof to show that vanilla Transformers are Turing-complete and then we prove that Transformers with only positional masking and without any positional encoding are also Turing-complete. We further analyze the necessity of each component for the Turing-completeness of the network; interestingly, we find that a particular type of residual connection is necessary. We demonstrate the practical implications of our results via experiments on machine translation and synthetic tasks.

1 Introduction

Transformer (Vaswani et al., 2017) is a recent self-attention based sequence-to-sequence architecture which has led to state of the art results across various NLP tasks including machine translation (Ott et al., 2018), language modeling (Radford et al., 2018) and question answering (Devlin et al., 2019). Although a number of variants of Transformers have been proposed, the original architecture still underlies these variants.

While the training and generalization of machine learning models such as Transformers are the central goals in their analysis, an essential prerequisite to this end is characterization of the computational power of the model: training a model for a certain task cannot succeed if the model is computationally incapable of carrying out the task. While the computational capabilities of recurrent networks (RNNs) have been studied for decades (Kolen and Kremer, 2001; Siegelmann, 2012), for Transformers we are still in the early stages.

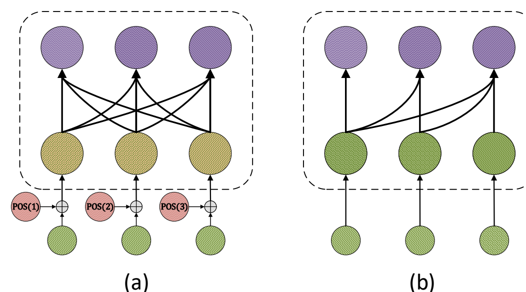

Figure 1: (a) Self-Attention Network with positional encoding, (b) Self-Attention Network with positional masking without any positional encoding

The celebrated work of Siegelmann and Sontag (1992) showed, assuming arbitrary precision, that RNNs are *Turing-complete*, meaning that they are capable of carrying out any algorithmic task formalized by Turing machines. Recently, Pérez et al. (2019) have shown that vanilla Transformers with hard-attention can also simulate Turing machines given arbitrary precision. However, in contrast to RNNs, Transformers consist of several components and it is unclear which components are necessary for its Turing-completeness and thereby crucial to its computational expressiveness.

The role of various components of the Transformer in its efficacy is an important question for further improvements. Since the Transformer does not process the input sequentially, it requires some form of positional information. Various positional encoding schemes have been proposed to capture order information (Shaw et al., 2018; Dai et al., 2019; Huang et al., 2018). At the same time, on

Proceedings of the 24th Conference on Computational Natural Language Learning, pages 455–475
Online, November 19-20, 2020. ©2020 Association for Computational Linguistics
https://doi.org/10.18653/v1/P17

machine translation, Yang et al. (2019) showed that the performance of Transformers with only positional masking (Shen et al., 2018) is comparable to that with positional encodings. In case of positional masking (Fig. 1), as opposed to explicit encodings, the model is only allowed to attend over preceding inputs and no additional positional encoding vector is combined with the input vector. Tsai et al. (2019) raised the question of whether explicit encoding is necessary if positional masking is used. Additionally, since Pérez et al. (2019)'s Turing-completeness proof relied heavily on residual connections, they asked whether these connections are essential for Turing-completeness. In this paper, we take a step towards answering such questions. Below, we list the main contributions of the paper,

- We provide an alternate and arguably simpler proof to show that Transformers are Turing-complete by directly relating them to RNNs.
- More importantly, we prove that Transformers with positional masking and without positional encoding are also Turing-complete.
- We analyze the necessity of various components such as self-attention blocks, residual connections and feedforward networks for Turing-completeness. Figure 2 provides an overview.
- We explore implications of our results on machine translation and synthetic tasks.[1]

2 Related Work

Computational Power of neural networks has been studied since the foundational paper McCulloch and Pitts (1943); in particular, among sequence-to-sequence models, this aspect of RNNs has long been studied (Kolen and Kremer, 2001). The seminal work by Siegelmann and Sontag (1992) showed that RNNs can simulate a Turing machine by using unbounded precision. Chen et al. (2018) showed that RNNs with ReLU activations are also Turing-complete. Many recent works have explored the computational power of RNNs in practical settings. Several works (Merrill et al., 2020), (Weiss et al., 2018) recently studied the ability of RNNs to recognize counter-like languages. The capability of RNNs to recognize strings of balanced parantheses has also been studied (Sennhauser and Berwick, 2018; Skachkova et al., 2018). However, such analysis on Transformers has been scarce.

Theoretical work on Transformers was initiated by Pérez et al. (2019) who formalized the notion of Transformers and showed that it can simulate a Turing machine given arbitrary precision. Concurrent to our work, there have been several efforts to understand self-attention based models (Levine et al., 2020; Kim et al., 2020). Hron et al. (2020) show that Transformers behave as Gaussian processes when the number of heads tend to infinity. Hahn (2020) showed some limitations of Transformer encoders in modeling regular and context-free languages. It has been recently shown that Transformers are universal approximators of sequence-to-sequence functions given arbitrary precision (Yun et al., 2020). However, these are not applicable[2] to the complete Transformer architecture. With a goal similar to ours, Tsai et al. (2019) attempted to study the attention mechanism via a kernel formulation. However, a systematic study of various components of Transformers has not been done.

3 Definitions and Preliminaries

All the numbers used in our computations will be from the set of rational numbers denoted $\mathbb{Q}$. For a sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$, we set $\boldsymbol{X}_j := (\boldsymbol{x}_1, \dots, \boldsymbol{x}_j)$ for $1 \leq j \leq n$. We will work with an alphabet Σ of size m, with special symbols $\#$ and $\$$ signifying the beginning and end of the input sequence, respectively. The symbols are mapped to vectors via a given 'base' embedding $f_b : \Sigma \to \mathbb{Q}^{d_b}$, where d_b is the dimension of the embedding. E.g., this embedding could be the one used for processing the symbols by the RNN.

We set $f_b(\#) = \mathbf{0}_{d_b}$ and $f_b(\$) = \mathbf{0}_{d_b}$. *Positional encoding* is a function $\text{pos} : \mathbb{N} \to \mathbb{Q}^{d_b}$. Together, these provide embedding for a symbol s at position i given by $f(f_b(s), \text{pos}(i))$, often taken to be simply $f_b(s) + \text{pos}(i)$. Vector $[\![s]\!] \in \mathbb{Q}^m$ denotes one-hot encoding of a symbol $s \in \Sigma$.

3.1 RNNs

We follow Siegelmann and Sontag (1992) in our definition of RNNs. To feed the sequences

[1]We have made our source code available at https://github.com/satwik77/Transformer-Computation-Analysis.

[2]Hahn (2020) and Yun et al. (2020) study encoder-only seq-to-seq models with fixed length outputs in which the computation halts as soon as the last symbol of the input is processed. Our work is about the full Transformer (encoder and decoder) which is a seq-to-seq model with variable length sequence output in which the decoder starts operating sequentially after the encoder.

$s_1 s_2 \dots s_n \in \Sigma^*$ to the RNN, these are converted to the vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n$ where $\boldsymbol{x}_i = f_b(s_i)$. The RNN is given by the recurrence $\boldsymbol{h}_t = g(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b})$, where $t \geq 1$, function $g(\cdot)$ is a multilayer feedforward network (FFN) with activation σ, bias vector $\boldsymbol{b} \in \mathbb{Q}^{d_h}$, matrices $\boldsymbol{W}_h \in \mathbb{Q}^{d_h \times d_h}$ and $\boldsymbol{W}_x \in \mathbb{Q}^{d_h \times d_b}$, and $\boldsymbol{h}_t \in \mathbb{Q}^{d_h}$ is the hidden state with given initial hidden state $\boldsymbol{h}_0$; d_h is the hidden state dimension.

After the last symbol s_n has been fed, we continue to feed the RNN with the terminal symbol $f_b(\$)$ until it halts. This allows the RNN to carry out computation after having read the input.

A class of seq-to-seq neural networks is Turing-complete if the class of languages recognized by the networks is exactly the class of languages recognized by Turing machines.

Theorem 3.1. *(Siegelmann and Sontag, 1992) Any seq-to-seq function $\Sigma^* \rightarrow \Sigma^*$ computable by a Turing machine can also be computed by an RNN.*

For details please see section B.1 in appendix.

3.2 Transformer Architecture

Vanilla Transformer. We describe the original Transformer architecture with positional encoding (Vaswani et al., 2017) as formalized by Pérez et al. (2019), with some modifications. All vectors in this subsection are from $\mathbb{Q}^d$.

The transformer, denoted Trans, is a seq-to-seq architecture. Its input consists of (i) a sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ of vectors, (ii) a seed vector $\boldsymbol{y}_0$. The output is a sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \dots, \boldsymbol{y}_r)$ of vectors. The sequence $\boldsymbol{X}$ is obtained from the sequence $(s_1, \dots, s_n) \in \Sigma^n$ of symbols by using the embedding mentioned earlier: $\boldsymbol{x}_i = f(f_b(s_i), \mathrm{pos}(i))$.

The transformer consists of composition of *transformer encoder* and *transformer decoder*. For the feedforward networks in the transformer layers we use the activation as in Siegelmann and Sontag (1992), namely the saturated linear activation function $\sigma(x)$ which takes value 0 for $x < 0$, value x for $0 < x < 1$ and value 1 for $x > 1$. This activation can be easily replaced by the standard ReLU activation via $\sigma(x) = \mathrm{ReLU}(x) - \mathrm{ReLU}(x-1)$.

Self-attention. The self-attention mechanism takes as input (i) a *query* vector $\boldsymbol{q}$, (ii) a sequence of *key* vectors $\boldsymbol{K} = (\boldsymbol{k}_1, \dots, \boldsymbol{k}_n)$, and (iii) a sequence of *value* vectors $\boldsymbol{V} = (\boldsymbol{v}_1, \dots, \boldsymbol{v}_n)$. The $\boldsymbol{q}$-attention over $\boldsymbol{K}$ and $\boldsymbol{V}$, denoted $\mathrm{Att}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V})$, is a vector $\boldsymbol{a} = \alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_n \boldsymbol{v}_n$, where (i) $(\alpha_1, \dots, \alpha_n) = \rho(f^{\mathrm{att}}(\boldsymbol{q}, \boldsymbol{k}_1), \dots, f^{\mathrm{att}}(\boldsymbol{q}, \boldsymbol{k}_n))$.
(ii) The normalization function $\rho : \mathbb{Q}^n \rightarrow \mathbb{Q}^n_{\geq 0}$ is $\mathsf{hardmax}$: for $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{Q}^n$, if the maximum value occurs r times among $x_1, \dots, x_n$, then $\mathsf{hardmax}(\boldsymbol{x})_i := 1/r$ if x_i is a maximum value and $\mathsf{hardmax}(\boldsymbol{x})_i := 0$ otherwise. In practice, the softmax is often used but its output values are in general not rational.
(iii) For vanilla transformers, the scoring function f^{att} used is a combination of multiplicative attention (Vaswani et al., 2017) and a non-linear function: $f^{\mathrm{att}}(\boldsymbol{q}, \boldsymbol{k}_i) = -\left|\langle \boldsymbol{q}, \boldsymbol{k}_i \rangle\right|$. This was also used by Pérez et al. (2019).

Transformer encoder. A *single-layer encoder* is a function $\mathrm{Enc}(\boldsymbol{X}; \boldsymbol{\theta})$, with input $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ a sequence of vectors in $\mathbb{Q}^d$, and parameters $\boldsymbol{\theta}$. The output is another sequence $\boldsymbol{Z} = (\boldsymbol{z}_1, \dots, \boldsymbol{z}_n)$ of vectors in $\mathbb{Q}^d$. The parameters $\boldsymbol{\theta}$ specify functions $Q(\cdot), K(\cdot), V(\cdot)$, and $O(\cdot)$, all of type $\mathbb{Q}^d \rightarrow \mathbb{Q}^d$. The functions $Q(\cdot), K(\cdot)$, and $V(\cdot)$ are linear transformations and $O(\cdot)$ an FFN. For $1 \leq i \leq n$, the output of the self-attention block is produced by

$$\boldsymbol{a}_i = \mathrm{Att}(Q(\boldsymbol{x}_i), K(\boldsymbol{X}), V(\boldsymbol{X})) + \boldsymbol{x}_i \quad (1)$$

This operation is also referred to as the encoder-encoder attention block. The output $\boldsymbol{Z}$ is computed by $\boldsymbol{z}_i = O(\boldsymbol{a}_i) + \boldsymbol{a}_i$ for $1 \leq i \leq n$. The addition operations $+\boldsymbol{x}_i$ and $+\boldsymbol{a}_i$ are the residual connections. The complete L-layer transformer encoder $\mathrm{TEnc}^{(L)}(\boldsymbol{X}; \boldsymbol{\theta}) = (\boldsymbol{K}^e, \boldsymbol{V}^e)$ has the same input $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ as the single-layer encoder. In contrast, its output $\boldsymbol{K}^e = (\boldsymbol{k}^e_1, \dots, \boldsymbol{k}^e_n)$ and $\boldsymbol{V}^e = (\boldsymbol{v}^e_1, \dots \boldsymbol{v}^e_n)$ contains two sequences. $\mathrm{TEnc}^{(L)}$ is obtained by composition of L single-layer encoders: let $\boldsymbol{X}^{(0)} := \boldsymbol{X}$, and for $0 \leq \ell \leq L-1$, let $\boldsymbol{X}^{(\ell+1)} = \mathrm{Enc}(\boldsymbol{X}^{(\ell)}; \boldsymbol{\theta}_\ell)$ and finally, $\boldsymbol{K}^e = K^{(L)}(\boldsymbol{X}^{(L)}), \quad \boldsymbol{V}^e = V^{(L)}(\boldsymbol{X}^{(L)})$.

Transformer decoder. The input to a *single-layer decoder* is (i) $(\boldsymbol{K}^e, \boldsymbol{V}^e)$ output by the encoder, and (ii) sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \dots, \boldsymbol{y}_k)$ of vectors for $k \geq 1$. The output is another sequence $\boldsymbol{Z} = (\boldsymbol{z}_1, \dots, \boldsymbol{z}_k)$.

Similar to the single-layer encoder, a single-layer decoder is parameterized by functions $Q(\cdot), K(\cdot), V(\cdot)$ and $O(\cdot)$ and is defined by

$$\boldsymbol{p}_t = \mathrm{Att}(Q(\boldsymbol{y}_t), K(\boldsymbol{Y}_t), V(\boldsymbol{Y}_t)) + \boldsymbol{y}_t, \quad (2)$$

$$\boldsymbol{a}_t = \mathrm{Att}(\boldsymbol{p}_t, \boldsymbol{K}^e, \boldsymbol{V}^e) + \boldsymbol{p}_t, \quad (3)$$

$$\boldsymbol{z}_t = O(\boldsymbol{a}_t) + \boldsymbol{a}_t,$$

where $1 \leq t \leq k$. The operation in (2) will be

referred to as the *decoder-decoder attention* block and the operation in (3) as the *decoder-encoder attention* block. In (2), positional masking is applied to prevent the network from attending over symbols which are ahead of them.

An L-layer Transformer decoder $\mathrm{TDec}^L((\boldsymbol{K}^e, \boldsymbol{V}^e), \boldsymbol{Y}; \boldsymbol{\theta}) = \boldsymbol{z}$ is obtained by repeated application of L single-layer decoders each with its own parameters, and a transformation function $F : \mathbb{Q}^d \to \mathbb{Q}^d$ applied to the last vector in the sequence of vectors output by the final decoder. Formally, for $0 \leq \ell \leq L-1$ and $\boldsymbol{Y}^0 := \boldsymbol{Y}$ we have $\boldsymbol{Y}^{\ell+1} = \mathrm{Dec}((\boldsymbol{K}^e, \boldsymbol{V}^e), \boldsymbol{Y}^\ell; \boldsymbol{\theta}_\ell)$, $\boldsymbol{z} = F(\boldsymbol{y}_k^L)$. Note that while the output of a single-layer decoder is a sequence of vectors, the output of an L-layer Transformer decoder is a single vector.

The complete Transformer. The output $\mathrm{Trans}(\boldsymbol{X}, \boldsymbol{y}_0) = \boldsymbol{Y}$ is computed by the recurrence $\tilde{\boldsymbol{y}}_{t+1} = \mathrm{TDec}(\mathrm{TEnc}(\boldsymbol{X}), (\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_t))$, for $0 \leq t \leq r-1$. We get $\boldsymbol{y}_{t+1}$ by adding positional encoding: $\boldsymbol{y}_{t+1} = \tilde{\boldsymbol{y}}_{t+1} + \mathrm{pos}(t+1)$.

Directional Transformer. We denote the Transformer with only positional masking and no positional encodings as Directional Transformer and use them interchangeably. In this case, we use standard multiplicative attention as the scoring function in our construction, i.e, $f^{\mathrm{att}}(\boldsymbol{q}, \boldsymbol{k}_i) = \langle \boldsymbol{q}, \boldsymbol{k}_i \rangle$. The general architecture is the same as for the vanilla case; the differences due to positional masking are the following.

There are no positional encodings. So the input vectors $\boldsymbol{x}_i$ only involve $f_b(s_i)$. Similarly, $\boldsymbol{y}_t = \tilde{\boldsymbol{y}}_t$. In (1), $\mathrm{Att}(\cdot)$ is replaced by $\mathrm{Att}(Q(\boldsymbol{x}_i), K(\boldsymbol{X}_i), V(\boldsymbol{X}_i))$ where $\boldsymbol{X}_i := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i)$ for $1 \leq i \leq n$. Similarly, in (3), $\mathrm{Att}(\cdot)$ is replaced by $\mathrm{Att}(\boldsymbol{p}_t, \boldsymbol{K}_t^e, \boldsymbol{V}_t^e)$.

Remark 1. Our definitions deviate slightly from practice, hard-attention being the main one since hardmax keeps the values rational whereas softmax takes the values to irrational space. Previous studies have shown that soft-attention behaves like hard-attention in practice and Hahn (2020) discusses its practical relevance.

Remark 2. Transformer Networks with positional encodings are not necessarily equivalent in terms of their computational expressiveness (Yun et al., 2020) to those with only positional masking when considering the encoder only model (as used in BERT and GPT-2). Our results in Section 4.1 show their equivalence in terms of expressiveness for the complete seq-to-seq architecture.

4 Primary Results

4.1 Turing-Completeness Results

In light of Theorem 3.1, to prove that Transformers are Turing-complete, it suffices to show that they can *simulate* RNNs. We say that a Transformer simulates an RNN (as defined in Sec. 3.1) if on every input $s \in \Sigma^*$, at each step t, the vector $\boldsymbol{y}_t$ contains the hidden state $\boldsymbol{h}_t$ as a subvector, i.e. $\boldsymbol{y}_t = [\boldsymbol{h}_t, \cdot]$, and halts at the same step as the RNN.

Theorem 4.1. *The class of Transformers with positional encodings is Turing-complete.*

Proof Sketch. The input $s_0, \ldots, s_n \in \Sigma^*$ is provided to the transformer as the sequence of vectors $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_n$, where $\boldsymbol{x}_i = [\boldsymbol{0}_{d_h}, f_b(s_i), \boldsymbol{0}_{d_h}, i, 1]$, which has as sub-vector the given base embedding $f_b(s_i)$ and the positional encoding i, along with extra coordinates set to constant values and will be used later.

The basic observation behind our construction of the simulating Transformer is that the transformer decoder can naturally implement the recurrence operations of the type used by RNNs. To this end, the FFN $O^{\mathrm{dec}}(\cdot)$ of the decoder, which plays the same role as the FFN component of the RNN, needs sequential access to the input in the same way as RNN. But the Transformer receives the whole input at the same time. We utilize positional encoding along with the attention mechanism to isolate $\boldsymbol{x}_t$ at time t and feed it to $O^{\mathrm{dec}}(\cdot)$, thereby simulating the RNN.

As stated earlier, we append the input $s_1, \ldots, s_n$ of the RNN with $'s until it halts. Since the Transformer takes its input all at once, appending by $'s is not possible (in particular, we do not know how long the computation would take). Instead, we append the input with a single $. After encountering a $ once, the Transformer will feed (encoding of) $ to $O^{\mathrm{dec}}(\cdot)$ in subsequent steps until termination. Here we confine our discussion to the case $t \leq n$; the $t > n$ case is slightly different but simpler.

The construction is straightforward: it has only one head, one encoder layer and one decoder layer; moreover, the attention mechanisms in the encoder and the decoder-decoder attention block of the decoder are trivial as described below.

The encoder attention layer does trivial computation in that it merely computes the identity function: $\boldsymbol{z}_i = \boldsymbol{x}_i$, which can be easily achieved, e.g. by using the residual connection and setting the value vectors to $\boldsymbol{0}$. The fi-

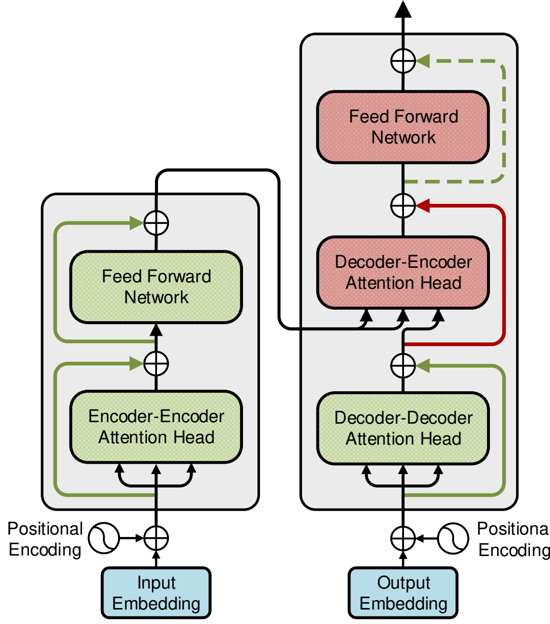

Figure 2: Transformer network with various components highlighted. The components marked red are essential for the Turing-completeness whereas for the pairs of blocks and residual connections marked green, either one of the component is enough. The dashed residual connection is not necessary for Turing-completeness of the network.

nal $K^{(1)}(\cdot)$ and $V^{(1)}(\cdot)$ functions bring $(\boldsymbol{K}^e, \boldsymbol{V}^e)$ into useful forms by appropriate linear transformations: $\boldsymbol{k}_i = [\mathbf{0}_{d_b}, \mathbf{0}_{d_b}, \mathbf{0}_{d_b}, -1, i]$ and $\boldsymbol{v}_i = [\mathbf{0}_{d_b}, f_b(s_i), \mathbf{0}_{d_b}, 0, 0]$. Thus, the key vectors only encode the positional information and the value vectors only encode the input symbols.

The output sequence of the decoder is $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots$. Our construction will ensure, by induction on t, that $\boldsymbol{y}_t$ contains the hidden states $\boldsymbol{h}_t$ of the RNN as a sub-vector along with positional information: $\boldsymbol{y}_t = [\boldsymbol{h}_t, \mathbf{0}_{d_b}, \mathbf{0}_{d_b}, t+1, 1]$. This is easy to arrange for $t = 0$, and assuming it for t we prove it for $t+1$. As for the encoder, the decoder-decoder attention block acts as the identity: $\boldsymbol{p}_t = \boldsymbol{y}_t$. Now, using the last but one coordinate in $\boldsymbol{y}_t$ representing the time $t+1$, the attention mechanism $\mathrm{Att}(\boldsymbol{p}_t, \boldsymbol{K}^e, \boldsymbol{V}^e)$ can retrieve the embedding of the t-th input symbol $\boldsymbol{x}_t$. This is possible because in the key vector $\boldsymbol{k}_i$ mentioned above, almost all coordinates other than the one representing the position i are set to 0, allowing the mechanism to only focus on the positional information and not be distracted by the other contents of $\boldsymbol{p}_t = \boldsymbol{y}_t$: the scoring function has value $f^{\mathrm{att}}(\boldsymbol{p}_t, \boldsymbol{k}_i) = -|\langle \boldsymbol{p}_t, \boldsymbol{k}_i \rangle| = -|i - (t+1)|$. For a given t, it is maximized at $i = t+1$ for $t < n$ and at $i = n$ for $t \geq n$. This use of scoring function is similar to Pérez et al. (2019).

At this point, $O^{\mathrm{dec}}(\cdot)$ has at its disposal the hidden state $\boldsymbol{h}_t$ (coming from $\boldsymbol{y}_t$ via $\boldsymbol{p}_t$ and the residual connection) and the input symbol $\boldsymbol{x}_t$ (coming via the attention mechanism and the residual connection). Hence $O(\cdot)$ can act just like the FFN (Lemma C.4) underlying the RNN to compute $\boldsymbol{h}_{t+1}$ and thus $\boldsymbol{y}_{t+1}$, proving the induction hypothesis. The complete construction can be found in Sec. C.2 in the appendix.

□

Theorem 4.2. *The class of Transformers with positional masking and no explicit positional encodings is Turing-complete.*

Proof Sketch. As before, by Theorem 3.1 it suffices to show that Transformers can simulate RNNs. The input $s_0, \ldots, s_n$ is provided to the transformer as the sequence of vectors $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_n$, where $\boldsymbol{x}_i = [\mathbf{0}_{d_h}, \mathbf{0}_{d_h}, f_b(s_i), [\![s_i]\!], 0, \mathbf{0}_m, \mathbf{0}_m, \mathbf{0}_m]$. The general goal for the directional case is similar to the vanilla case, namely we would like the FFN $O^{\mathrm{dec}}(\cdot)$ of the decoder to directly simulate the computation in the underlying RNN. In the vanilla case, positional encoding and the attention mechanism helped us feed input $\boldsymbol{x}_t$ at the t-th iteration of the decoder to $O^{\mathrm{dec}}(\cdot)$. However, we no longer have explicit positional information in the input $\boldsymbol{x}_t$ such as a coordinate with value t. The key insight is that we do not need the positional information explicitly to recover $\boldsymbol{x}_t$ at step t: in our construction, the attention mechanism with masking will recover $\boldsymbol{x}_t$ in an indirect manner even though it's not able to "zero in" on the t-th position.

Let us first explain this without details of the construction. We maintain in vector $\boldsymbol{\omega}_t \in \mathbb{Q}^m$, with a coordinate each for symbols in Σ, the fraction of times the symbol has occurred up to step t. Now, at a step $t \leq n$, for the difference $\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1}$ (which is part of the query vector), it can be shown easily that only the coordinate corresponding to s_t is positive. Thus after applying the linearized sigmoid $\sigma(\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1})$, we can isolate the coordinate corresponding to s_t. Now using this query vector, the (hard) attention mechanism will be able to retrieve the value vectors for all indices j such that $s_j = s_t$ and output their average. Crucially, the value vector for an index j is essentially $\boldsymbol{x}_j$ which depends only on s_j. Thus, all these vectors are equal to $\boldsymbol{x}_t$, and so is their average. This recovers $\boldsymbol{x}_t$, which

can now be fed to $O^{\mathrm{dec}}(\cdot)$, simulating the RNN.

We now outline the construction and relate it to the above discussion. As before, for simplicity we restrict to the case $t \leq n$. We use only one head, one layer encoder and two layer decoder. The encoder, as in the vanilla case, does very little other than pass information along. The vectors in $(\boldsymbol{K}^e, \boldsymbol{V}^e)$ are obtained by the trivial attention mechanism followed by simple linear transformations: $\boldsymbol{k}_i^e = [\mathbf{0}_{d_h}, \mathbf{0}_{d_h}, \mathbf{0}_{d_b}, [\![s_i]\!], 0, \mathbf{0}_m, \mathbf{0}_m, \mathbf{0}_m]$ and $\boldsymbol{v}_i^e = [\mathbf{0}_{d_h}, \mathbf{0}_{d_h}, f_b(s_i), \mathbf{0}_m, 0, \mathbf{0}_m, [\![s_i]\!], \mathbf{0}_m]$.

Our construction ensures that at step t we have $\boldsymbol{y}_t = [\boldsymbol{h}_{t-1}, \mathbf{0}_{d_h}, \mathbf{0}_{d_b}, \mathbf{0}_m, \frac{1}{2^t}, \mathbf{0}_m, \mathbf{0}_m, \boldsymbol{\omega}_{t-1}]$. As before, the proof is by induction on t.

In the first layer of decoder, the decoder-decoder attention block is trivial: $\boldsymbol{p}_t^{(1)} = \boldsymbol{y}_t$. In the decoder-encoder attention block, we give equal attention to all the $t+1$ values, which along with $O^{\mathrm{enc}}(\cdot)$, leads to $\boldsymbol{z}_t^{(1)} = [\boldsymbol{h}_{t-1}, \ \mathbf{0}_{d_h}, \ \mathbf{0}_{d_b}, \boldsymbol{\delta}_t, \ \frac{1}{2^{t+1}}, \mathbf{0}_m, \mathbf{0}_m, \ \boldsymbol{\omega}_t]$, where essentially $\boldsymbol{\delta}_t = \sigma(\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1})$, except with a change for the last coordinate due to special status of the last symbol \$ in the processing of RNN.

In the second layer, the decoder-decoder attention block is again trivial with $\boldsymbol{p}_t^{(2)} = \boldsymbol{z}_t^{(1)}$. We remark that in this construction, the scoring function is the standard multiplicative attention [3]. Now $\langle \boldsymbol{p}_t^{(2)}, \boldsymbol{k}_j^e \rangle = \langle \boldsymbol{\delta}_t, [\![s_j]\!] \rangle = \boldsymbol{\delta}_{t,j}$, which is positive if and only if $s_j = s_t$, as mentioned earlier. Thus attention weights in $\mathrm{Att}(\boldsymbol{p}_t^{(2)}, \boldsymbol{K}_t^e, \boldsymbol{V}_t^e)$ satisfy $\mathsf{hardmax}(\langle \boldsymbol{p}_t^{(2)}, \boldsymbol{k}_1^e \rangle, \ldots, \langle \boldsymbol{p}_t^{(2)}, \boldsymbol{k}_t^e \rangle) = \frac{1}{\lambda_t}(\mathbb{I}(s_0 = s_t), \mathbb{I}(s_1 = s_t), \ldots, \mathbb{I}(s_t = s_t))$, where λ_t is a normalization constant and $\mathbb{I}(\cdot)$ is the indicator. See Lemma D.3 for more details.

At this point, $O^{\mathrm{dec}}(\cdot)$ has at its disposal the hidden state $\boldsymbol{h}_t$ (coming from $\boldsymbol{z}_t^{(1)}$ via $\boldsymbol{p}_t^{(2)}$ and the residual connection) and the input symbol $\boldsymbol{x}_t$ (coming via the attention mechanism and the residual connection). Hence $O^{\mathrm{dec}}(\cdot)$ can act just like the FFN underlying the RNN to compute $\boldsymbol{h}_{t+1}$ and thus $\boldsymbol{y}_{t+1}$, proving the induction hypothesis.

The complete construction can be found in Sec. D in the Appendix.

□

In practice, Yang et al. (2019) found that for NMT, Transformers with only positional masking achieve comparable performance compared to the ones with positional encodings. Similar evidence was found by Tsai et al. (2019). Our proof for directional transformers entails that there is no loss of order information if positional information is only provided in the form of masking. However, we do not recommend using masking as a replacement for explicit encodings. The computational equivalence of encoding and masking given by our results implies that any differences in their performance must come from differences in learning dynamics.

4.2 Analysis of Components

The results for various components follow from our construction in Theorem 4.1. Note that in both the encoder and decoder attention blocks, we need to compute the identity function. We can nullify the role of the attention heads by setting the value vectors to zero and making use of only the residual connections to implement the identity function. Thus, even if we remove those attention heads, the model is still Turing-complete. On the other hand, we can remove the residual connections around the attention blocks and make use of the attention heads to implement the identity function by using positional encodings. Hence, either the attention head or the residual connection is sufficient to achieve Turing-completeness. A similar argument can be made for the FFN in the encoder layer: either the residual connection or the FFN is sufficient for Turing-completeness. For the decoder-encoder attention head, since it is the only way for the decoder to obtain information about the input, it is necessary for the completeness. The FFN is the only component that can perform computations based on the input and the computations performed earlier via recurrence and hence, the model is not Turing-complete without it. Figure 2 summarizes the role of different components with respect to the computational expressiveness of the network.

Proposition 4.3. *The class of Transformers without residual connection around the decoder-encoder attention block is not Turing-complete.*

Proof Sketch. We confine our discussion to single-layer decoder; the case of multilayer decoder is similar. Without the residual connection, the decoder-encoder attention block produces $\boldsymbol{a}_t = \mathrm{Att}(\boldsymbol{p}_t, \boldsymbol{K}^e, \boldsymbol{V}^e) = \sum_{i=1}^n \alpha_i \boldsymbol{v}_i^e$ for some α_i's such that $\sum_i^n \alpha_i = 1$. Note that, without residual connection $\boldsymbol{a}_t$ can take on at most $2^n - 1$ values. This is because by the definition of hard attention the vector $(\alpha_1, \ldots, \alpha_n)$ is characterized by the set of zero coordinates and there are at most $2^n - 1$

[3]Note that it is closer to practice than the scoring function $-|\langle \boldsymbol{q}, \boldsymbol{k} \rangle|$ used in Pérez et al. (2019) and Theorem 4.1

such sets (all coordinates cannot be zero). This restriction on the number of values on $\boldsymbol{a}_t$ holds regardless of the value of $\boldsymbol{p}_t$. If the task requires the network to produce values of $\boldsymbol{a}_t$ that come from a set with size at least 2^n, then the network will not be able to perform the task. Here's an example task: given a number $\Delta \in (0, 1)$, the network must produce numbers $0, \Delta, 2\Delta, \ldots, k\Delta$, where k is the maximum integer such that $k\Delta \leq 1$. If the network receives a single input Δ, then it is easy to see that the vector $\boldsymbol{a}_t$ will be a constant ($\boldsymbol{v}_1^e$) at any step and hence the output of the network will also be constant at all steps. Thus, the model cannot perform such a task. If the input is combined with $n - 1$ auxiliary symbols (such as # and $), then in the network, each $\boldsymbol{a}_t$ takes on at most $2^n - 1$ values. Hence, the model will be incapable of performing the task if $\Delta < 1/2^n$. Such a limitation does not exist with a residual connection since the vector $\boldsymbol{a}_t = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i^e + \boldsymbol{p}_t$ can take arbitrary number of values depending on its prior computations in $\boldsymbol{p}_t$. For further details, see Sec. C.1 in the Appendix. □

Discussion. It is perhaps surprising that residual connection, originally proposed to assist in the learning ability of very deep networks, plays a vital role in the computational expressiveness of the network. Without it, the model is limited in its capability to make decisions based on predictions in the previous steps. We explore practical implications of this result in section 5.

5 Experiments

In this section, we explore the practical implications of our results. Our experiments are geared towards answering the following questions:

Q1. Are there any practical implications of the limitation of Transformers without decoder-encoder residual connections? What tasks can they do or not do compared to vanilla Transformers?

Q2. Is there any additional benefit of using positional masking as opposed to absolute positional encoding (Vaswani et al., 2017)?

Although we showed that Transformers without decoder-encoder residual connection are not Turing complete, it does not imply that they are incapable of performing all the tasks. Our results suggest that they are limited in their capability to make inferences based on their previous computations, which is required for tasks such as counting and language modeling. However, it can be shown that the model is capable of performing tasks which rely only on information provided at a given step such as copying and mapping. For such tasks, given positional information at a particular step, the model can look up the corresponding input and map it via the FFN. We evaluate these hypotheses via our experiments.

Model	**Copy Task**	**Counting**
Vanilla Transformers	100.0	100.0
- Dec-Enc Residual	99.7	0.0
- Dec-Dec Residual	99.7	99.8

Table 1: BLEU scores (↑) for copy and counting task. Please see Section 5 for details

For our experiments on synthetic data, we consider two tasks, namely the *copy task* and the *counting task*. For the copy task, the goal of a model is to reproduce the input sequence. We sample sentences of lengths between 5-12 words from Penn Treebank and create a train-test split of 40k-1k with all sentences belonging to the same range of length. In the counting task, we create a very simple dataset where the model is given one number between 0 and 100 as input and its goal is to predict the next five numbers. Since only a single input is provided to the encoder, it is necessary for the decoder to be able to make inferences based on its previous predictions to perform this task. The benefit of conducting these experiments on synthetic data is that they isolate the phenomena we wish to evaluate. For both these tasks, we compare vanilla Transformer with the one without decoder-encoder residual connection. As a baseline we also consider the model without decoder-decoder residual connection, since according to our results, that connection does not influence the computational power of the model. We implement a single layer encoder-decoder network with only a single attention head in each block.

We then assess the influence of the limitation on Machine Translation which requires a model to do a combination of both mapping and inferring from computations in previous timesteps. We evaluate the models on IWSLT'14 German-English dataset and IWSLT'15 English-Vietnamese dataset. We again compare vanilla Transformer with the ones without decoder-encoder and decoder-decoder residual connection. While tuning the models, we vary the number of layers from 1 to 4, the learning rate, warmup steps and the number of heads. Specifications of the models, experimental setup, datasets and sample outputs can be found in Sec. E

Model	De-En	En-Vi
Vanilla Transformers	32.9	28.8
- Dec-Enc Residual	24.1	21.8
- Dec-Dec Residual	30.6	27.2

Table 2: BLEU scores (↑) for translation task. Please see Section 5 for details.

in the Appendix.

Results on the effect of residual connections on synthetic tasks can be found in Table 1. As per our hypothesis, all the variants are able to perfectly perform the copy task. For the counting task, the one without decoder-encoder residual connection is incapable of performing it. However, the other two including the one without decoder-decoder residual connection are able to accomplish the task by learning to make decisions based on their prior predictions. Table 3 provides some illustrative sample outputs of the models. For the MT task, results can be found in Table 2. While the drop from removing decoder-encoder residual connection is significant, it is still able to perform reasonably well since the task can be largely fulfilled by mapping different words from one sentence to another.

For positional masking, our proof technique suggests that due to lack of positional encodings, the model must come up with its own mechanism to make order related decisions. Our hypothesis is that, if it is able to develop such a mechanism, it should be able to generalize to higher lengths and not overfit on the data it is provided. To evaluate this claim, we simply extend the copy task upto higher lengths. The training set remains the same as before, containing sentences of length 5-12 words. We create 5 different validation sets each containing 1k sentences each. The first set contains sentences within the same length as seen in training (5-12 words), the second set contains sentences of length 13-15 words while the third, fourth and fifth sets contain sentences of lengths 15-20, 21-25 and 26-30 words respectively. We consider two models, one which is provided absolute positional encodings and one where only positional masking is applied. Figure 3 shows the performance of these models across various lengths. The model with positional masking clearly generalizes up to higher lengths although its performance too degrades at extreme lengths. We found that the model with absolute positional encodings during training overfits on the fact that the 13th token is always the terminal symbol. Hence, when evaluated on higher lengths it never produces a sentence of length greater than 12. Other encoding schemes such as relative positional encodings (Shaw et al., 2018; Dai et al., 2019) can generalize better, since they are inherently designed to address this particular issue. However, our goal is not to propose masking as a replacement of positional encodings, rather it is to determine whether the mechanism that the model develops during training is helpful in generalizing to higher lengths. Note that, positional masking was not devised by keeping generalization or any other benefit in mind. Our claim is only that, the use of masking does not limit the model's expressiveness and it may benefit in other ways, but during practice one should explore each of the mechanisms and even a combination of both. Yang et al. (2019) showed that a combination of both masking and encodings is better able to learn order information as compared to explicit encodings.

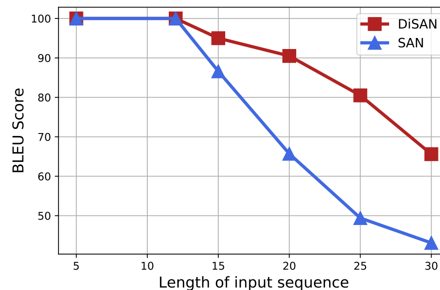

Figure 3: Performance of the two models on the copy task across varying lengths of test inputs. DiSAN refers to Transformer with only positional masking. SAN refers to vanilla Transformers.

SOURCE	– 42
REFERENCE	– 43 44 45 46 47
VANILLA TRANSFORMER	– 43 44 45 46 47
- DEC-ENC RESIDUAL	– 27 27 27 27 27
- DEC-DEC RESIDUAL	– 43 44 45 46 47

Table 3: Sample outputs by the models on the counting task. Without the residual connection around Decoder-Encoder block, the model is incapable of predicting more than one distinct output.

6 Discussion and Final Remarks

We showed that the class of languages recognized by Transformers and RNNs are exactly the same. This implies that the difference in performance of both the networks across different tasks can be attributed only to their learning abilities. In contrast to RNNs, Transformers are composed of multiple components which are not essential for their com-

putational expressiveness. However, in practice they may play a crucial role. Recently, Voita et al. (2019) showed that the decoder-decoder attention heads in the lower layers of the decoder do play a significant role in the NMT task and suggest that they may be helping in language modeling. This indicates that components which are not essential for the computational power may play a vital role in improving the learning and generalization ability.

Take-Home Messages. We showed that the order information can be provided either in the form of explicit encodings or masking without affecting computational power of Transformers. The decoder-encoder attention block plays a necessary role in conditioning the computation on the input sequence while the residual connection around it is necessary to keep track of previous computations. The feedforward network in the decoder is the only component capable of performing computations based on the input and prior computations. Our experimental results show that removing components essential for computational power inhibit the model's ability to perform certain tasks. At the same time, the components which do not play a role in the computational power may be vital to the learning ability of the network.

Although our proofs rely on arbitrary precision, which is common practice while studying the computational power of neural networks in theory (Siegelmann and Sontag, 1992; Pérez et al., 2019; Hahn, 2020; Yun et al., 2020), implementations in practice work over fixed precision settings. However, our construction provides a starting point to analyze Transformers under finite precision. Since RNNs can recognize all regular languages in finite precision (Korsky and Berwick, 2019), it follows from our construction that Transformer can also recognize a large class of regular languages in finite precision. At the same time, it does not imply that it can recognize all regular languages given the limitation due to the precision required to encode positional information. We leave the study of Transformers in finite precision for future work.

Acknowledgements

We thank the anonymous reviewers for their constructive comments and suggestions. We would also like to thank our colleagues at Microsoft Research and Michael Hahn for their valuable feedback and helpful discussions.

References

Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May, and Kevin Knight. 2018. Recurrent neural networks as weighted language recognizers. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2261–2271, New Orleans, Louisiana. Association for Computational Linguistics.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171.

Jiri Hron, Yasaman Bahri, Jascha Sohl-Dickstein, and Roman Novak. 2020. Infinite attention: Nngp and ntk for deep attention networks. *arXiv preprint arXiv:2006.10540*.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, and Douglas Eck. 2018. An improved relative self-attention mechanism for transformer with application to music generation. *ArXiv*, abs/1809.04281.

Hyunjik Kim, George Papamakarios, and Andriy Mnih. 2020. The lipschitz constant of self-attention. *arXiv preprint arXiv:2006.04710*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada. Association for Computational Linguistics.

John F Kolen and Stefan C Kremer. 2001. *A field guide to dynamical recurrent networks*. John Wiley & Sons.

Samuel A Korsky and Robert C Berwick. 2019. On the computational power of rnns. *arXiv preprint arXiv:1906.06349*.

Yoav Levine, Noam Wies, Or Sharir, Hofit Bata, and Amnon Shashua. 2020. Limits to depth efficiencies of self-attention. *arXiv preprint arXiv:2006.12467*.

Minh-Thang Luong and Christopher D Manning. 2015. Stanford neural machine translation systems for spoken language domains. In *Proceedings of the International Workshop on Spoken Language Translation*, pages 76–79.

Warren S McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. 2020. A formal hierarchy of RNN architectures. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 443–459, Online. Association for Computational Linguistics.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9, Brussels, Belgium. Association for Computational Linguistics.

Jorge Pérez, Javier Marinković, and Pablo Barceló. 2019. On the turing completeness of modern neural network architectures. In *International Conference on Learning Representations*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*.

Alexander Rush. 2018. The annotated transformer. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 52–60, Melbourne, Australia. Association for Computational Linguistics.

Luzi Sennhauser and Robert Berwick. 2018. Evaluating the ability of LSTMs to learn context-free grammars. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 115–124, Brussels, Belgium. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. 2018. Disan: Directional self-attention network for rnn/cnn-free language understanding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Hava T Siegelmann. 2012. *Neural networks and analog computation: beyond the Turing limit*. Springer Science & Business Media.

Hava T Siegelmann and Eduardo D Sontag. 1992. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449. ACM.

Natalia Skachkova, Thomas Trost, and Dietrich Klakow. 2018. Closing brackets with recurrent neural networks. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 232–239, Brussels, Belgium. Association for Computational Linguistics.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2019. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4344–4353, Hong Kong, China. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy. Association for Computational Linguistics.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745, Melbourne, Australia. Association for Computational Linguistics.

Baosong Yang, Longyue Wang, Derek F. Wong, Lidia S. Chao, and Zhaopeng Tu. 2019. Assessing the ability of self-attention networks to learn word order. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3635–3644, Florence, Italy. Association for Computational Linguistics.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. 2020. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*.

A Roadmap

We begin with various definitions and results. We define simulation of Turing machines by RNNs and state the Turing-completeness result for RNNs. We define vanilla and directional Transformers and what it means for Transformers to simulate RNNs. Many of the definitions from the main paper are reproduced here, but in more detail. In Sec. C.1 we discuss the effect of removing a residual connection on computational power of Transformers. Sec. C.2 contains the proof of Turing completeness of vanilla Transformers and Sec. D the corresponding proof for directional Transformers. Finally, Sec. 5 has further details of experiments.

B Definitions

Denote the set $\{1, 2, \ldots, n\}$ by $[n]$. Functions defined for scalars are extended to vectors in the natural way: for a function F defined on a set A, for a sequence $(a_1, \ldots, a_n)$ of elements in A, we set $F(a_1, \ldots, a_n) := (F(a_1), \ldots, F(a_n))$. Indicator $\mathbb{I}(P)$ is 1, if predicate P is true and is 0 otherwise. For a sequence $\boldsymbol{X} = (\boldsymbol{x}_{n'}, \ldots, \boldsymbol{x}_n)$ for some $n' \geq 0$, we set $\boldsymbol{X}_j := (\boldsymbol{x}_{n'}, \ldots, \boldsymbol{x}_j)$ for $j \in \{n', i+1, \ldots, n\}$. We will work with an alphabet $\Sigma = \{\beta_1, \ldots, \beta_m\}$, with $\beta_1 = \#$ and $\beta_m = \$$. The special symbols $\#$ and $\$$ correspond to the beginning and end of the input sequence, resp. For a vector $\boldsymbol{v}$, by $\mathbf{0}_{\boldsymbol{v}}$ we mean the all-0 vector of the same dimension as $\boldsymbol{v}$. Let $\bar{t} := \min\{t, n\}$

B.1 RNNs and Turing-completeness

Here we summarize, somewhat informally, the Turing-completeness result for RNNs due to (Siegelmann and Sontag, 1992). We recall basic notions from computability theory. In the main paper, for simplicity we stated the results for *total recursive* functions $\phi : \{0,1\}^* \to \{0,1\}^*$, i.e. a function that is defined on every $s \in \{0,1\}^*$ and whose values can be computed by a Turing machine. While total recursive functions form a satisfactory formalization of seq-to-seq tasks, here we state the more general result for *partial recursive functions*. Let $\phi : \{0,1\}^* \to \{0,1\}^*$ be partial recursive. A partial recursive function is one that need not be defined for every $s \in \{0,1\}^*$, and there exists a Turing Machine $\mathcal{M}$ with the following property. The input s is initially written on the tape of the Turing Machine $\mathcal{M}$ and the output $\phi(s)$ is the content of the tape upon acceptance which is indicated by halting in a designated accept state. On s for which ϕ is undefined, $\mathcal{M}$ does not halt.

We now specify how Turing machine $\mathcal{M}$ is simulated by RNN $R(\mathcal{M})$. In the RNNs in (Siegelmann and Sontag, 1992) the hidden state $\boldsymbol{h}_t$ has the form

$$\boldsymbol{h}_t = [\boldsymbol{q}_t, \Psi_1, \Psi_2],$$

where $\boldsymbol{q}_t = [q_1, \ldots, q_s]$ denotes the state of $\mathcal{M}$ one-hot form. Numbers $\Psi_1, \Psi_2 \in \mathbb{Q}$, called stacks, store the contents of the tape in a certain Cantor set like encoding (which is similar to, but slightly more involved, than binary representation) at each step. The simulating RNN $R(\mathcal{M})$, gets as input encodings of $s_1 s_2 ... s_n$ in the first n steps, and from then on receives the vector $\mathbf{0}$ as input in each step. If ϕ is defined on s, then $\mathcal{M}$ halts and accepts with the output $\phi(s)$ the content of the tape. In this case, $R(\mathcal{M})$ enters a special accept state, and Ψ_1 encodes $\phi(s)$ and $\Psi_2 = 0$. If $\mathcal{M}$ does not halt then $R(\mathcal{M})$ also does not enter the accept state.

Siegelmann and Sontag (1992) further show that from $R(\mathcal{M})$ one can further explicitly produce the $\phi(s)$ as its output. In the present paper, we will not deal with explicit production of the output but rather work with the definition of simulation in the previous paragraph. This is for simplicity of exposition, and the main ideas are already contained in our results. If the Turing machine computes $\phi(s)$ in time $T(s)$, the simulation takes $O(|s|)$ time to encode the input sequence s and $4T(s)$ to compute $\phi(s)$.

Theorem B.1 ((Siegelmann and Sontag, 1992))**.** *Given any partial recursive function $\phi : \{0,1\}^* \to \{0,1\}^*$ computed by Turing machine $\mathcal{M}_\phi$, there exists a simulating RNN $R(\mathcal{M}_\phi)$.*

In view of the above theorem, for establishing Turing-completeness of Transformers, it suffices to show that RNNs can be simulated by Transformers. Thus, in the sequel we will only talk about simulating RNNs.

B.2 Vanilla Transformer Architecture

Here we describe the original transformer architecture due to (Vaswani et al., 2017) as formalized by (Pérez et al., 2019). While our notation and definitions largely follow (Pérez et al., 2019), they are not identical. The transformer here makes use of positional encoding; later we will discuss the transformer variant using directional attention but without using positional encoding.

The transformer, denoted Trans, is a sequence-to-sequence architecture. Its input consists of (i) a sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ of vectors in $\mathbb{Q}^d$, (ii) a seed vector $\boldsymbol{y}_0 \in \mathbb{Q}^d$. The output is a sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \dots, \boldsymbol{y}_r)$ of vectors in $\mathbb{Q}^d$. The sequence $\boldsymbol{X}$ is obtained from the sequence $(s_0, \dots, s_n) \in \Sigma^{n+1}$ of symbols by using the embedding mentioned earlier: $\boldsymbol{x}_i = f(f_b(s_i), \text{pos}(i))$ for $0 \leq i \leq n$. The transformer consists of composition of *transformer encoder* and a *transformer decoder*. The transformer encoder is obtained by composing one or more *single-layer encoders* and similarly the transformer decoder is obtained by composing one or more *single-layer decoders*. For the feed-forward networks in the transformer layers we use the activation as in (Siegelmann and Sontag, 1992), namely the saturated linear activation function:

$$\sigma(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } 0 \leq x \leq 1, \\ 1 & \text{if } x > 1. \end{cases} \tag{4}$$

As mentioned in the main paper, we can easily work with the standard ReLU activation via $\sigma(x) = \mathsf{ReLU}(x) - \mathsf{ReLU}(x-1)$. In the following, after defining these components, we will put them together to specify the full transformer architecture. But we begin with self-attention mechanism which is the central feature of the transformer.

Self-attention. The self-attention mechanism takes as input (i) a *query* vector $\boldsymbol{q}$, (ii) a sequence of *key* vectors $\boldsymbol{K} = (\boldsymbol{k}_1, \dots, \boldsymbol{k}_n)$, and (iii) a sequence of *value* vectors $\boldsymbol{V} = (\boldsymbol{v}_1, \dots, \boldsymbol{v}_n)$. All vectors are in $\mathbb{Q}^d$.

The $\boldsymbol{q}$-attention over keys $\boldsymbol{K}$ and values $\boldsymbol{V}$, denoted by $\text{Att}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V})$, is a vector $\boldsymbol{a}$ given by

$$\begin{aligned} (\alpha_1, \dots, \alpha_n) &= \rho(f^{\text{att}}(\boldsymbol{q}, \boldsymbol{k}_1), \dots, f^{\text{att}}(\boldsymbol{q}, \boldsymbol{k}_n)), \\ \boldsymbol{a} &= \alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_n \boldsymbol{v}_n. \end{aligned}$$

The above definition uses two functions ρ and f^{att} which we now describe. For the normalization function $\rho : \mathbb{Q}^n \to \mathbb{Q}^n_{\geq 0}$ we will use hardmax: for $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{Q}^n$, if the maximum value occurs r times among $x_1, \dots, x_n$, then $\mathsf{hardmax}(\boldsymbol{x})_i := 1/r$ if x_i is a maximum value and $\mathsf{hardmax}(\boldsymbol{x})_i := 0$ otherwise. In practice, the softmax is often used but its output values are in general not rational. The names soft-attention and hard-attention are used for the attention mechanism depending on which normalization function is used.

For the Turing-completeness proof of vanilla transformers, the scoring function f^{att} used is a combination of multiplicative attention (Vaswani et al., 2017) and a non-linear function: $f^{\text{att}}(\boldsymbol{q}, \boldsymbol{k}_i) = -\left|\langle \boldsymbol{q}, \boldsymbol{k}_i \rangle\right|$. For directional transformers, the standard multiplicative attention is used, that is, $f^{\text{att}}(\boldsymbol{q}, \boldsymbol{k}_i) = \langle \boldsymbol{q}, \boldsymbol{k}_i \rangle$.

Transformer encoder. A *single-layer encoder* is a function $\text{Enc}(\boldsymbol{X}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the parameter vector and the input $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ is a sequence of vector in $\mathbb{Q}^d$. The output is another sequence $\boldsymbol{Z} = (\boldsymbol{z}_1, \dots, \boldsymbol{z}_n)$ of vectors in $\mathbb{Q}^d$. The parameters $\boldsymbol{\theta}$ specify functions $Q(\cdot), K(\cdot), V(\cdot)$, and $O(\cdot)$, all of type $\mathbb{Q}^d \to \mathbb{Q}^d$. The functions $Q(\cdot), K(\cdot)$, and $V(\cdot)$ are usually linear transformations and this will be the case in our constructions:

$$\begin{aligned} Q(x_i) &= \boldsymbol{x}_i^T W_Q, \\ K(x_i) &= \boldsymbol{x}_i^T W_K, \\ V(x_i) &= \boldsymbol{x}_i^T W_V, \end{aligned}$$

where $W_Q, W_K, W_V \in \mathbb{Q}^{d \times d}$. The function $O(\cdot)$ is a feed-forward network. The single-layer encoder is then defined by

$$\begin{aligned} \boldsymbol{a}_i &= \text{Att}(Q(\boldsymbol{x}_i), K(\boldsymbol{X}), V(\boldsymbol{X})) + \boldsymbol{x}_i, \\ \boldsymbol{z}_i &= O(\boldsymbol{a}_i) + \boldsymbol{a}_i. \end{aligned} \tag{5}$$

The addition operations $+\boldsymbol{x}_i$ and $+\boldsymbol{a}_i$ are the residual connections. The operation in (5) is called the encoder-encoder attention block.

The complete L-layer transformer encoder $\text{TEnc}^{(L)}(\boldsymbol{X}; \boldsymbol{\theta})$ has the same input $\boldsymbol{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$ as the single-layer encoder. By contrast, its output consists of two sequences $(\boldsymbol{K}^e, \boldsymbol{V}^e)$, each a sequence of n vectors in $\mathbb{Q}^d$. The encoder $\text{TEnc}^{(L)}(\cdot)$ is obtained by repeated application of single-layer encoders, each with its own parameters; and at the end, two trasformation functions $K^L(\cdot)$ and $V^L(\cdot)$ are applied to the sequence of output vectors at the last layer. Functions $K^{(L)}(\cdot)$ and $V^{(L)}(\cdot)$ are linear transformations in our constructions. Formally, for $1 \leq \ell \leq L-1$ and $\boldsymbol{X}^1 := \boldsymbol{X}$, we have

$$\begin{aligned} \boldsymbol{X}^{\ell+1} &= \text{Enc}(\boldsymbol{X}^\ell; \boldsymbol{\theta}_\ell), \\ \boldsymbol{K}^e &= K^{(L)}(\boldsymbol{X}^L), \\ \boldsymbol{V}^e &= V^{(L)}(\boldsymbol{X}^L). \end{aligned}$$

The output of the L-layer Transformer encoder $(\boldsymbol{K}^e, \boldsymbol{V}^e) = \text{TEnc}^{(L)}(\boldsymbol{X})$ is fed to the Transformer decoder which we describe next.

Transformer decoder. The input to a *single-layer decoder* is (i) $(\boldsymbol{K}^e, \boldsymbol{V}^e)$, the sequences of key and value vectors output by the encoder, and (ii) a sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_k)$ of vectors in $\mathbb{Q}^d$. The output is another sequence $\boldsymbol{Z} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k)$ of vectors in $\mathbb{Q}^d$.

Similar to the single-layer encoder, a single-layer decoder is parameterized by functions $Q(\cdot), K(\cdot), V(\cdot)$ and $O(\cdot)$ and is defined by

$$\begin{aligned} \boldsymbol{p}_t &= \text{Att}(Q(\boldsymbol{y}_t), K(\boldsymbol{Y}_t), V(\boldsymbol{Y}_t)) + \boldsymbol{y}_t, \quad (6) \\ \boldsymbol{a}_t &= \text{Att}(\boldsymbol{p}_t, \boldsymbol{K}^e, \boldsymbol{V}^e) + \boldsymbol{p}_t, \quad (7) \\ \boldsymbol{z}_t &= O(\boldsymbol{a}_t) + \boldsymbol{a}_t. \end{aligned}$$

The operation in (6) will be referred to as the *decoder-decoder attention* block and the operation in (7) as the *decoder-encoder attention* block. In the decoder-decoder attention block, positional masking is applied to prevent the network from attending over symbols which are ahead of them.

An L-layer Transformer decoder is obtained by repeated application of L single-layer decoders each with its own parameters and a transformation function $F : \mathbb{Q}^d \to \mathbb{Q}^d$ applied to the last vector in the sequence of vectors output by the final decoder. Formally, for $1 \leq \ell \leq L-1$ and $\boldsymbol{Y}^1 = \boldsymbol{Y}$ we have

$$\begin{aligned} \boldsymbol{Y}^{\ell+1} &= \text{Dec}((\boldsymbol{K}^e, \boldsymbol{V}^e), \boldsymbol{Y}^\ell; \boldsymbol{\theta}_\ell), \\ \boldsymbol{z} &= F(\boldsymbol{y}_t^L). \end{aligned}$$

We use $\boldsymbol{z} = \text{TDec}^L((\boldsymbol{K}^e, \boldsymbol{V}^e), \boldsymbol{Y}; \boldsymbol{\theta})$ to denote an L-layer Transformer decoder. Note that while the output of a single-layer decoder is a sequence of vectors, the output of an L-layer Transformer decoder is a single vector.

The complete Transformer. A *Transformer network* receives an input sequence $\boldsymbol{X}$, a seed vector $\boldsymbol{y}_0$, and $r \in \mathbb{N}$. For $t \geq 0$ its output is a sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_r)$ defined by

$$\tilde{\boldsymbol{y}}_{t+1} = \text{TDec}\left(\text{TEnc}(\boldsymbol{X}), (\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_t)\right).$$

We get $\boldsymbol{y}_{t+1}$ by adding positional encoding: $\boldsymbol{y}_{t+1} = \tilde{\boldsymbol{y}}_{t+1} + \text{pos}(t+1)$. We denote the complete Transformer by $\text{Trans}(\boldsymbol{X}, \boldsymbol{y}_0) = \boldsymbol{Y}$. The Transformer "halts" when $\boldsymbol{y}_T \in H$, where H is a prespecified halting set.

Simulation of RNNs by Transformers. We say that a Transformer simulates an RNN (as defined in Sec. B.1) if on input $s \in \Sigma^*$, at each step t, the vector $\boldsymbol{y}_t$ contains the hidden state $\boldsymbol{h}_t$ as a subvector: $\boldsymbol{y}_t = [\boldsymbol{h}_t, \cdot]$, and halts at the same step as RNN.

C Results on Vanilla Transformers

C.1 Residual Connections

Proposition C.1. *The Transformer without residual connection around the Decoder-Encoder Attention block in the Decoder is not Turing Complete*

Proof. Recall that the vectors $\boldsymbol{a}_t$ is produced from the Encoder-Decoder Attention block in the following way,

$$\boldsymbol{a}_t = \text{Att}(\boldsymbol{p}_t, \boldsymbol{K}^e, \boldsymbol{V}^e) + \boldsymbol{p}_t$$

The result follows from the observation that without the residual connections, $\boldsymbol{a}_t = \text{Att}(\boldsymbol{p}_t, \boldsymbol{K}^e, \boldsymbol{V}^e)$, which leads to $\boldsymbol{a}_t = \sum_{i=1}^n \alpha_i \boldsymbol{v}_i^e$ for some α_is such that $\sum_i^n \alpha_i = 1$. Since $\boldsymbol{v}_i^e$ is produced from the encoder, the vector $\boldsymbol{a}_t$ will have no information about its previous hidden state values. Since the previous hidden state information was computed and stored in $\boldsymbol{p}_t$, without the residual connection, the information in $\boldsymbol{a}_t$ depends solely on the output of the encoder.

One could argue that since the attention weights α_is depend on the query vector $\boldsymbol{p}_t$, it could still use it gain the necessary information from the vectors $\boldsymbol{v}_i^e$s. However, note that by definition of hard attention, the attention weights α_i in $\boldsymbol{a}_t = \sum_{i=1}^n \alpha_i \boldsymbol{v}_i^e$ can either be zero or some nonzero value depending on the attention logits. Since the attention weights α_i are such that $\sum_i^n \alpha_i = 1$ and all the nonzero weights are equal to each other. Thus given the constraints there are $2^n - 1$ ways to attend over n inputs excluding the case where no input is attended over. Hence, the network without decoder-encoder residual connection with n inputs can have at most $2^n - 1$ distinct $\boldsymbol{a}_t$ values. This implies that the model will be unable to perform a task that takes n inputs and has to produce more than $2^n - 1$ outputs. Note that, such a limitation will not exist with a residual connection since the vector $\boldsymbol{a}_t = \Sigma_{i=1}^n \alpha_i \boldsymbol{v}_i^e + \boldsymbol{p}_t$ can take arbitrary number of values depending on its prior computations in $\boldsymbol{p}_t$.

As an example to illustrate the limitation, consider the following simple problem, given a value Δ, where $0 \leq \Delta \leq 1$, the network must produce

the values $0, \Delta, 2\Delta, \ldots, k\Delta$, where k is the maximum integer such that $k\Delta \leq 1$. If the network receives a single input Δ, the encoder will produce only one particular output vector and regardless of what the value of the query vector $\boldsymbol{p}_t$ is, the vector $\boldsymbol{a}_t$ will be constant at every timestep. Since $\boldsymbol{a}_t$ is fed to feedforward network which maps it to $\boldsymbol{z}_t$, the output of the decoder will remain the same at every timestep and it cannot produce distinct values. If the input is combined with $n-1$ auxiliary symbols (such as # and $), then the network can only produce $2^n - 1$ outputs. Hence, the model will be incapable of performing the task if $\Delta < 1/2^n$.

Thus the model cannot perform the task defined above which RNNs and Vanilla Transformers can easily do with a simple counting mechanism via their recurrent connection.

For the case of **multilayer decoder**, consider any L layer decoder model. If the residual connection is removed, the output of decoder-encoder attention block at each layer is $\boldsymbol{a}_t^{(\ell)} = \sum_{i=1}^{n} \alpha_i^{(\ell)} \boldsymbol{v}_i^e$ for $1 \leq \ell \leq L$. Observe, that since output of the decoder-encoder attention block in the last (L-th) layer of the decoder is $\boldsymbol{a}_t^{(L)} = \sum_{i=1}^{n} \alpha_i^{(L)} \boldsymbol{v}_i^e$. Since the output of the L layer decoder will be a feedforward network over $\boldsymbol{a}_t^{(L)}$, the computation reduces to the single layer decoder case. Hence, similar to the single layer case, if the task requires the network to produce values of $\boldsymbol{a}_t$ that come from a set with size at least 2^n, then the network will not be able to perform the task.

This implies that the model without decoder-encoder residual connection is limited in its capability to perform tasks which requires it to make inferences based on previously generated outputs. □

C.2 Simulation of RNNs by Transformers with positional encoding

Theorem C.2. *RNNs can be simulated by vanilla Transformers and hence the class of vanilla Transformers is Turing-complete.*

Proof. The construction of the simulating transformer is simple: it uses a single head and both the encoder and decoder have one layer. Moreover, the encoder does very little and most of the action happens in the decoder. The main task for the simulation is to design the input embedding (building on the given base embedding f_b), the feedforward network $O(\cdot)$ and the matrices corresponding to functions $Q(\cdot), K(\cdot), V(\cdot)$.

Input embedding. The input embedding is obtained by summing the symbol and positional encodings which we next describe. These encodings have dimension $d = 2d_h + d_b + 2$, where d_h is the dimension of the hidden state of the RNN and d_b is the dimension of the given encoding f_b of the input symbols. We will use the symbol encoding $f^{\text{symb}} : \Sigma \to \mathbb{Q}^d$ which is essentially the same as f_b except that the dimension is now larger:

$$f^{\text{symb}}(s) = [\mathbf{0}_{d_h}, f_e(s);\ \mathbf{0}_{d_h}, 0, 0].$$

The positional encoding $\text{pos} : \mathbb{N} \to \mathbb{Q}^d$ is simply

$$\text{pos}(i) = [\mathbf{0}_{d_h}, \mathbf{0}_{d_b}, \mathbf{0}_{d_h}, i, 1].$$

Together, these define the combined embedding f for a given input sequence $s_0 s_1 \cdots s_n \in \Sigma^*$ by

$$f(s_i) = f^{\text{symb}}(s_i) + \text{pos}(i) = [\mathbf{0}_{d_h}, f_b(s_i), \mathbf{0}_{d_h}, i, 1].$$

The vectors $\boldsymbol{v} \in \mathbb{Q}^d$ used in the computation of our transformer are of the form

$$\boldsymbol{v} = [\boldsymbol{h}_1, \boldsymbol{s};\ \boldsymbol{h}_2, x_1, x_2],$$

where $\boldsymbol{h}_1, \boldsymbol{h}_2 \in \mathbb{Q}^{d_h}$, $\boldsymbol{s} \in \mathbb{Q}^{d_e}$, and $x_1, x_2 \in \mathbb{Q}$. The coordinates corresponding to the $\boldsymbol{h}_i$'s are reserved for computation related to hidden states of the RNN, the coordinates corresponding to $\boldsymbol{s}$ are reserved for base embeddings, and those for x_1 and x_2 are reserved for scalar values related to positional operations. The first two blocks, corresponding to $\boldsymbol{h}_1$ and $\boldsymbol{s}$ are reserved for computation of the RNN.

During the computation of the Transformer, the underlying RNN will get the input $\boldsymbol{s}_{\bar{t}}$ at step t for $t = 0, 1, \ldots$, where recall that $\bar{t} = \min\{t, n\}$. This sequence leads to the RNN getting the embedding of the input sequence $s_0, \ldots, s_n$ in the first $n+1$ steps followed by the embedding of the symbol $ for the subsequent steps, which is in accordance with the requirements of (Siegelmann and Sontag, 1992). Similar to (Pérez et al., 2019) we use the following scoring function in the attention mechanism in our construction,

$$f^{\text{att}}(\boldsymbol{q}_i, \boldsymbol{k}_j) = -|\langle \boldsymbol{q}_i, \boldsymbol{k}_j \rangle| \tag{8}$$

Construction of TEnc**.** As previously mentioned, our transformer encoder has only one layer, and the computation in the encoder is very simple: the attention mechanism is not utilized, only the residual connections are. This is done by setting

the matrix for $V(\cdot)$ to the all-zeros matrix, and the feedforward networks to always output $\mathbf{0}$. The application of appropriately chosen linear transformations for the final $K(\cdot)$ and $V(\cdot)$ give the following lemma about the output of the encoder.

Lemma C.3. *There exists a single layer encoder denoted by* TEnc *that takes as input the sequence* $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \$)$ *and generates the tuple* $(\boldsymbol{K}^e, \boldsymbol{V}^e)$ *where* $\boldsymbol{K}^e = (\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n)$ *and* $\boldsymbol{V}^e = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n)$ *such that,*

$$\begin{aligned} \boldsymbol{k}_i &= [\mathbf{0}_h, \mathbf{0}_s;\ \mathbf{0}_h, -1, i], \\ \boldsymbol{v}_i &= [\mathbf{0}_h, \boldsymbol{s}_i;\ \mathbf{0}_h, 0, 0]. \end{aligned}$$

Construction of TDec**.** As in the construction of TEnc, our TDec has only one layer. Also like TEnc, the decoder-decoder attention block just computes the identity: we set $V^{(1)}(\cdot) = \mathbf{0}$ identically, and use the residual connection so that $\boldsymbol{p}_t = \boldsymbol{y}_t$.

For $t \geq 0$, at the t-th step we denote the input to the decoder as $\boldsymbol{y}_t = \tilde{\boldsymbol{y}}_t + \mathrm{pos}(t)$. Let $\boldsymbol{h}_0 = \mathbf{0}_h$ and $\tilde{\boldsymbol{y}}_0 = \mathbf{0}$. We will show by induction that at the t-th timestep we have

$$\boldsymbol{y}_t = [\boldsymbol{h}_t, \mathbf{0}_s;\ \mathbf{0}_h, t+1, 1]. \tag{9}$$

By construction, this is true for $t = 0$:

$$\boldsymbol{y}_0 = [\mathbf{0}_h, \mathbf{0}_s;\ \mathbf{0}_h, 1, 1].$$

Assuming that it holds for t, we show it for $t+1$.

By Lemma C.5

$$\mathrm{Att}(\boldsymbol{p}_t, \boldsymbol{K}^e, \boldsymbol{V}^e) = [\mathbf{0}_h, \boldsymbol{v}_{\overline{t+1}};\ \mathbf{0}_h, 0, 0]. \tag{10}$$

Lemma C.5 basically shows how we retrieve the input $\boldsymbol{s}_{\overline{t+1}}$ at the relevant step for further computation in the decoder. It follows that

$$\begin{aligned} \boldsymbol{a}_t &= \mathrm{Att}(\boldsymbol{p}_t, \boldsymbol{K}^e, \boldsymbol{V}^e) + \boldsymbol{p}_t \\ &= [\boldsymbol{h}_t, \boldsymbol{s}_{\overline{t+1}}, \mathbf{0}_h, t+1, 1]. \end{aligned}$$

In the final block of the decoder, the computation for RNN takes place:

Lemma C.4. *There exists a function* $O(\cdot)$ *defined by feed-forward network such that,*

$$O(\boldsymbol{a}_t) = [(\boldsymbol{h}_{t+1} - \boldsymbol{h}_t), -\boldsymbol{s}_{\overline{t+1}}, \mathbf{0}_h, -(t+1), -1],$$

where $\boldsymbol{W}_h, \boldsymbol{W}_x$ *and* $\boldsymbol{b}$ *denote the parameters of the RNN under consideration.*

This leads to

$$\boldsymbol{z}_t = O(\boldsymbol{a}_t) + \boldsymbol{a}_t = [\boldsymbol{h}_{t+1}, \mathbf{0}_s;\ \mathbf{0}_h, 0, 0].$$

□

We choose the function F for our decoder to be the identity function, therefore $\tilde{\boldsymbol{y}}_{t+1} = [\boldsymbol{h}_{t+1}, \mathbf{0}_s;\ \mathbf{0}_h, 0, 0]$, which means $\boldsymbol{y}_{t+1} = \tilde{\boldsymbol{y}}_{t+1} + \mathrm{pos}(i+1) = [\boldsymbol{h}_{t+1}, \mathbf{0}_s;\ \mathbf{0}_h, t+2, 1]$, proving our induction hypothesis.

C.3 Technical Lemmas

Proof of Lemma C.3. We construct a single-layer encoder achieving the desired $\boldsymbol{K}^e$ and $\boldsymbol{V}^e$. We make use of the residual connections and via trivial self-attention we get that $\boldsymbol{z}_i = \boldsymbol{x}_i$. More specifically for $i \in [n]$ we have

$$\begin{aligned} V^{(1)}(\boldsymbol{x}_i) &= \mathbf{0}, \\ \boldsymbol{a}_i &= \mathbf{0} + \boldsymbol{x}_i, \\ O(\boldsymbol{a}_i) &= \mathbf{0}, \\ \boldsymbol{z}_i &= \mathbf{0} + \boldsymbol{a}_i = \boldsymbol{x}_i. \end{aligned}$$

$V^{(1)}(\boldsymbol{x}_i) = \mathbf{0}$ can be achieved by setting the weight matrix as the all-0 matrix. Recall that $\boldsymbol{x}_i$ is defined as

$$\boldsymbol{x}_i = [\ \mathbf{0}_h, \boldsymbol{s}_i, \ \mathbf{0}_h, i, 1\ ].$$

We then apply linear transformations in $K(\boldsymbol{z}_i) = \boldsymbol{z}_i \boldsymbol{W}_k$ and $V(\boldsymbol{z}_i) = \boldsymbol{z}_i \boldsymbol{W}_v$, where

$$\boldsymbol{W}_k^T = \left[\begin{array}{ccccc} 0 & 0 & \cdots & 0 & 0 \\ \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ \hline 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & -1 & 0 \end{array}\right],$$

and $\boldsymbol{W}_k \in \mathbb{Q}^{d \times d}$, and similarly one can obtain $\boldsymbol{v}_i$ by setting the submatrix of $\boldsymbol{W}_v \in \mathbb{Q}^{d \times d}$ formed by the first $d-2$ rows and columns to the identity matrix, and the rest of the entries to zeros.

□

Lemma C.5. *Let* $\boldsymbol{q}_t \in \mathbb{Q}^d$ *be a query vector such that* $\boldsymbol{q} = [\cdot, \ldots, \cdot, t+1, 1]$ *where* $t \in \mathbb{N}$ *and '·' denotes an arbitrary value. Then we have*

$$\mathrm{Att}(\boldsymbol{q}_t, \boldsymbol{K}^e, \boldsymbol{V}^e) = [\mathbf{0}_h, \boldsymbol{s}_{\overline{t+1}}, \mathbf{0}_h, 0, 0]. \tag{11}$$

Proof. Recall that $\boldsymbol{p}_t = \boldsymbol{y}_t = [\boldsymbol{h}_t, 0, \ldots, 0, t+1, 1]$ and $\boldsymbol{k}_i = [0, 0, \ldots, 0, -1, i]$ and hence

$$\langle \boldsymbol{p}_t, \boldsymbol{k}_i \rangle = i - (t+1),$$

$$f^{\text{att}}(\boldsymbol{p}_t, \boldsymbol{k}_i) = -|i - (t+1)|.$$

Thus, for $i \in [n]$, the scoring function $f^{\text{att}}(\boldsymbol{p}_t, \boldsymbol{k}_i)$ has the maximum value 0 at index $i = t+1$ if $t < n$; for $t \geq n$, the maximum value $t + 1 - n$ is achieved for $i = n$. Therefore

$$\text{Att}(\boldsymbol{p}_t, \boldsymbol{K}^e, \boldsymbol{V}^e) = \boldsymbol{s}_{\overline{t+1}}.$$

□

Proof of Lemma C.4. Recall that

$$\boldsymbol{a}_t = [\ \boldsymbol{h}_t, \boldsymbol{s}_{\overline{t+1}}, \mathbf{0}_h, t+1, 1\ ]$$

Network $O(\boldsymbol{a}_t)$ is of the form

$$O(\boldsymbol{a}_t) = \mathbf{W}_2 \sigma(\mathbf{W}_1 \boldsymbol{a}_t + \boldsymbol{b}_1),$$

where $\mathbf{W}_i \in \mathbb{Q}^{d \times d}$ and $\boldsymbol{b} \in \mathbb{Q}^d$ and

$$\mathbf{W}_1 = \begin{array}{c} \\ d_h \\ d_e \\ d_h \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} d_h & d_e & d_h & 2 \end{array} \\ \left[\begin{array}{c|c|c|c} \mathbf{W}_h & \mathbf{W}_x & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{array}\right] \end{array}$$

and $\boldsymbol{b}_1 = [\boldsymbol{b}_h, \mathbf{0}_s, \mathbf{0}_h, 0, 0]$. Hence

$$\sigma(\mathbf{W}_1 \boldsymbol{a}_t + \boldsymbol{b}_1) = [\sigma(\mathbf{W}_h \boldsymbol{h}_t + \mathbf{W}_x \boldsymbol{s}_{\overline{t+1}} + \boldsymbol{b}), \boldsymbol{s}_{\overline{t+1}}, \boldsymbol{h}_t, t+1, 1]$$

Next we define $\mathbf{W}_2$ by

$$\mathbf{W}_2 = \begin{array}{c} \\ d_h \\ d_e \\ d_h \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} d_h & d_e & d_h & 2 \end{array} \\ \left[\begin{array}{c|c|c|c} \mathbf{I} & \mathbf{0} & -\mathbf{I} & \mathbf{0} \\ \hline \mathbf{0} & -\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & -\mathbf{I} \end{array}\right] \end{array}.$$

This leads to

$$\begin{aligned} O(\boldsymbol{a}_t) &= \mathbf{W}_2 \sigma(\mathbf{W}_1 \boldsymbol{a}_t + \boldsymbol{b}_1) \\ &= [\sigma(\mathbf{W}_h \boldsymbol{h}_t + \mathbf{W}_x \boldsymbol{s}_{\overline{t+1}} + \boldsymbol{b}) - \boldsymbol{h}_t, -\boldsymbol{s}_{\overline{t+1}}, \\ &\qquad \mathbf{0}_h, -(t+1), -1], \end{aligned}$$

which is what we wanted to prove. □

D Completeness of Directional Transformers

There are a few changes in the architecture of the Transformer to obtain directional Transformer. The first change is that there are no positional encodings and thus the input vector $\boldsymbol{x}_i$ only consists of $\boldsymbol{s}_i$. Similarly, there are no positional encodings in the decoder inputs and hence $\boldsymbol{y}_t = \tilde{\boldsymbol{y}}_t$. The vector $\tilde{\boldsymbol{y}}$ is the output representation produced at the previous step and the first input vector to the decoder $\tilde{\boldsymbol{y}}_0 = \mathbf{0}$. Instead of using positional encodings, we apply positional masking to the inputs and outputs of the encoder.

Thus the encoder-encoder attention in (5) is redefined as

$$\boldsymbol{a}_i^{(\ell+1)} = \text{Att}(Q(\boldsymbol{z}_i^{(\ell)}), K(\boldsymbol{Z}_i^{(\ell)}), V(\boldsymbol{Z}_i^{(\ell)})) + \boldsymbol{z}_i^{(\ell)},$$

where $\boldsymbol{Z}^{(0)} = \boldsymbol{X}$. Similarly the decoder-encoder attention in (7) is redefined by

$$\boldsymbol{a}_t^{(\ell)} = \text{Att}(\boldsymbol{p}_t^{(\ell)}, \boldsymbol{K}_t^e, \boldsymbol{V}_t^e) + \boldsymbol{p}_t^{(\ell)},$$

where ℓ in $\boldsymbol{a}_t^{(\ell)}$ denotes the layer ℓ and we use $\boldsymbol{v}^{(\ell,b)}$ to denote any intermediate vector being used in ℓ-th layer and b-th block in cases where the same symbol is used in multiple blocks in the same layer.

Theorem D.1. *RNNs can be simulated by vanilla Transformers and hence the class of vanilla Transformers is Turing-complete.*

Proof. The Transformer network in this case will be more complex than the construction for the vanilla case. The encoder remains very similar, but the decoder is different and has two layers.

Embedding. We will construct our Transformer to simulate an RNN of the form given in the definition with the recurrence

$$\boldsymbol{h}_t = g(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{b}).$$

The vectors used in the Transformer layers are of dimension $d = 2d_h + d_e + 4|\Sigma| + 1$. Where d_h is the dimension of the hidden state of the RNN and d_e is the dimension of the input embedding.

All vector $\boldsymbol{v} \in \mathbb{Q}^d$ used during the computation of the network are of the form

$$\boldsymbol{v} = [\boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{s}_1, [\![s_1]\!], x_1, [\![s_2]\!][\![s_3]\!], [\![s_4]\!]]$$

where $\boldsymbol{h}_i \in \mathbb{Q}^{d_h}, \boldsymbol{s} \in \mathbb{Q}^{d_e}$ and $x_i \in \mathbb{Q}$. These blocks reserved for different types of objects. The

vectors $\boldsymbol{h}_i$s are reserved for computation related to hidden states of RNNs, $\boldsymbol{s}_i$s are reserved for input embeddings and x_is are reserved for scalar values related to positional operations.

Given an input sequence $s_0 s_1 s_2 \cdots s_n \in \Sigma^*$ where $s_0 = \#$ and $s_n = \$$, we use an embedding function $f : \Sigma \to \mathbb{Q}^d$ defined as

$$f(s_i) = \boldsymbol{x}_i \quad = \quad [\quad \mathbf{0}_h, \mathbf{0}_h, \boldsymbol{s}_i, \\ [\![s_i]\!], 0, \mathbf{0}_\omega, \mathbf{0}_\omega, \mathbf{0}_\omega \quad]$$

Unlike (Pérez et al., 2019), we use the dot product as our scoring function as used in Vaswani et al. (2017) in the attention mechanism in our construction,

$$f^{\text{att}}(\boldsymbol{q}_i, \boldsymbol{k}_j) = \langle \boldsymbol{q}_i, \boldsymbol{k}_j \rangle.$$

For the computation of the Transformer, we also use a vector sequence in $\mathbb{Q}^{|\Sigma|}$ defined by

$$\boldsymbol{\omega}_t = \frac{1}{t+1} \sum_{j=0}^{t} [\![s_t]\!],$$

where $0 \leq t \leq n$. The vector $\boldsymbol{\omega}_t = (\boldsymbol{\omega}_{t,1}, \ldots, \boldsymbol{\omega}_{t,|\Sigma|})$ contains the proportion of each input symbol till step t for $0 \leq t \leq n$. Set $\boldsymbol{\omega}_{-1} = \mathbf{0}$. From the defintion of $\boldsymbol{\omega}_t$, it follows that at any step $1 \leq k \leq |\Sigma|$ we have

$$\boldsymbol{\omega}_{t,k} = \frac{\phi_{t,k}}{t+1}, \tag{12}$$

where $\phi_{t,k}$ denotes the number of times the k-th symbol β_k in Σ has appeared till the t-th step. Note that $\boldsymbol{\omega}_{t,0} = \frac{1}{t+1}$ since the first coordinate corresponds to the proportion of the start symbol $\#$ which appears only once at $t = 0$. Similarly, $\boldsymbol{\omega}_{t,|\Sigma|} = 0$ for $0 \leq t < n$ and $\boldsymbol{\omega}_{t,|\Sigma|} = 1/(t+1)$ for $t \geq n$, since the end symbol \$ doesn't appear till the end of the input and it appears only once at $t = n$.

We define two more sequences of vectors in $\mathbb{Q}^{|\Sigma|}$ for $0 \leq t \leq n$:

$$\boldsymbol{\Delta}_t = \sigma(\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1}), \\ \boldsymbol{\delta}_t = (\boldsymbol{\Delta}_{t,1}, \ldots, \boldsymbol{\Delta}_{t,|\Sigma|-1}, 1/2^{t+1}).$$

Here $\boldsymbol{\Delta}_t$ denotes the difference in the proportion of symbols between the t-th and $(t-1)$-th steps, with the applicatin of sigmoid activation. In vector $\boldsymbol{\delta}_t$, the last coordinate of $\boldsymbol{\Delta}_t$ has been replaced with $1/2^{t+1}$. The last coordinate in $\boldsymbol{\omega}_t$ indicates the proportion of the terminal symbol \$ and hence the last value in $\boldsymbol{\Delta}_t$ denotes the change in proportion of \$. We set the last coordinate in $\boldsymbol{\delta}_t$ to an exponentially decreasing sequence so that after n steps we always have a nonzero score for the terminal symbol and it is taken as input in the underlying RNN. Different and perhaps simpler choices for the last coordinate of $\boldsymbol{\delta}_t$ may be possible. Note that $0 \leq \boldsymbol{\Delta}_{t,k} \leq 1$ and $0 \leq \boldsymbol{\delta}_{t,k} \leq 1$ for $0 \leq t \leq n$ and $1 \leq k \leq |\Sigma|$.

Construction of TEnc**.** The input to the network DTrans_M is the sequence $(s_0, s_1, \ldots, s_{n-1}, s_n)$ where $s_0 = \#$ and $s_n = \$$. Our encoder is a simple single layer network such that $\mathrm{TEnc}(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = (\boldsymbol{K}^e, \boldsymbol{V}^e)$ where $\boldsymbol{K}^e = (\boldsymbol{k}^e_0, \ldots, \boldsymbol{k}^e_n)$ and $\boldsymbol{V}^e = (\boldsymbol{v}^e_0, \ldots, \boldsymbol{v}^e_n)$ such that,

$$\boldsymbol{k}^e_i \quad = \quad [\quad \mathbf{0}_h, \mathbf{0}_h, \mathbf{0}_s, \\ [\![s_i]\!], 0, \mathbf{0}_\omega, \mathbf{0}_\omega, \mathbf{0}_\omega \quad], \tag{13}$$

$$\boldsymbol{v}^e_i \quad = \quad [\quad \mathbf{0}_h, \mathbf{0}_h, \boldsymbol{s}_i, \\ \mathbf{0}_\omega, 0, \mathbf{0}_\omega, [\![s_i]\!], \mathbf{0}_\omega \quad].$$

Similar to our construction of the encoder for vanilla transformer (Lemma C.3), the above $\boldsymbol{K}^e$ and $\boldsymbol{V}^e$ can be obtained by making the output of $\mathrm{Att}(\cdot) = 0$ by choosing the $V(\cdot)$ to always evaluate to 0 and similarly for $O(\cdot)$, and using residual connections. Then one can produce $\boldsymbol{K}^e$ and $\boldsymbol{V}^e$ via simple linear transformations using $K(\cdot)$ and $V(\cdot)$.

Construction of TDec**.** At the t-th step we denote the input to the decoder as $\boldsymbol{y}_t = \tilde{\boldsymbol{y}}_t$, where $0 \leq t \leq r$, where r is the step where the decoder halts. Let $\boldsymbol{h}_{-1} = \mathbf{0}_h$ and $\boldsymbol{h}_0 = \mathbf{0}_h$. We will prove by induction on t that for $0 \leq t \leq r$ we have

$$\boldsymbol{y}_t \quad = \quad [\quad \boldsymbol{h}_{t-1}, \mathbf{0}_h, \mathbf{0}_s, \\ \mathbf{0}_\omega, \tfrac{1}{2^t}, \mathbf{0}_\omega, \mathbf{0}_\omega, \boldsymbol{\omega}_{\overline{t-1}} \quad]. \tag{14}$$

This is true for $t = 0$ by the choice of seed vector:

$$\boldsymbol{y}_0 \quad = \quad [\quad \mathbf{0}_h, \mathbf{0}_h, \mathbf{0}_s, \\ \mathbf{0}_\omega, 1, \mathbf{0}_\omega, \mathbf{0}_\omega, \mathbf{0}_\omega \quad].$$

Assuming the truth of (14) for t, we show it for $t + 1$.

Layer 1. Similar to the construction in Lemma C.3, in the decoder-decoder attention block we set $V^{(1)}(\cdot) = \mathbf{0}_d$ and use the residual connections to set $\boldsymbol{p}^{(1)}_t = \boldsymbol{y}_t$. At the t-th step in the decoder-encoder attention block of layer 1 we have

$$\mathrm{Att}(\boldsymbol{p}^{(1)}_t, \boldsymbol{K}^e_{\bar{t}}, \boldsymbol{V}^e_{\bar{t}}) = \sum_{j=0}^{\bar{t}} \hat{\alpha}^{(1,2)}_{t,j} \boldsymbol{v}^e_j,$$

where

$$(\hat{\alpha}^{(2,2)}_{t,1}, \ldots, \hat{\alpha}^{(2,2)}_{t,\bar{t}}) = \mathsf{hardmax}\left(\langle \boldsymbol{p}^{(1)}_t, \boldsymbol{k}^e_1\rangle, \ldots, \langle \boldsymbol{p}^{(1)}_t, \boldsymbol{k}^e_{\bar{t}}\rangle\right) = \mathsf{hardmax}(0, \ldots, 0) = \left(\frac{1}{\bar{t}+1}, \ldots, \frac{1}{\bar{t}+1}\right).$$

Therefore

$$\textstyle\sum_{j=0}^{\bar{t}} \hat{\alpha}^{(1,2)}_{t,j} \boldsymbol{v}^e_j = [\ \mathbf{0}_h, \mathbf{0}_h, \boldsymbol{s}_{0:t}, \mathbf{0}_\omega, 0, \mathbf{0}_\omega, \boldsymbol{\omega}_{\bar{t}}, \mathbf{0}_\omega\ ]$$

where

$$\boldsymbol{s}_{0:t} = \frac{1}{\bar{t}+1} \sum_{j=0}^{\bar{t}} \boldsymbol{s}_j.$$

Thus,

$$\boldsymbol{a}^{(1)}_t = \mathrm{Att}(\boldsymbol{p}^{(1)}_t, \boldsymbol{K}^e_{\bar{t}}, \boldsymbol{V}^e_{\bar{t}}) + \boldsymbol{p}^{(1)}_t = [\boldsymbol{h}_{t-1}, \mathbf{0}_h, \boldsymbol{s}_{0:t}, \mathbf{0}_\omega, \frac{1}{2^t}, \mathbf{0}_\omega, \boldsymbol{\omega}_{\bar{t}}, \boldsymbol{\omega}_{\overline{t-1}}].$$

In Lemma D.2 we construct feed-forward network $O^{(1)}(\cdot)$ such that

$$O^{(1)}(\boldsymbol{a}^{(1)}_t) = [\mathbf{0}_h, \mathbf{0}_h, -\boldsymbol{s}_{0:t}, \boldsymbol{\delta}_{\bar{t}}, -\frac{1}{2^t} + \frac{1}{2^{t+1}}, \mathbf{0}_\omega, -\boldsymbol{\omega}_{\bar{t}}, -\boldsymbol{\omega}_{\overline{t-1}} + \boldsymbol{\omega}_{\bar{t}}].$$

Hence

$$\boldsymbol{z}^{(1)}_t = O^{(1)}(\boldsymbol{a}^{(1)}_t) + \boldsymbol{a}^{(1)}_t = [\boldsymbol{h}_{t-1}, \mathbf{0}_h, \mathbf{0}_s, \boldsymbol{\delta}_{\bar{t}}, \frac{1}{2^{t+1}}, \mathbf{0}_\omega, \mathbf{0}_\omega, \boldsymbol{\omega}_{\bar{t}}]. \tag{15}$$

Layer 2. In the first block of layer 2, we set the value transformation function to identically zero similar to Lemma C.3, i.e. $V^{(2)}(\cdot) = \mathbf{0}$ which leads to the output of $\mathrm{Att}(\cdot)$ to be $\mathbf{0}$ and then using the residual connection we get $\boldsymbol{p}^{(2)}_t = \boldsymbol{z}^{(1)}_t$. It follows by Lemma D.3 that

$$\mathrm{Att}(\boldsymbol{p}^{(2)}_t, \boldsymbol{K}^e_{\bar{t}}, \boldsymbol{V}^e_{\bar{t}}) = [\mathbf{0}_h, \mathbf{0}_h, \boldsymbol{s}_{\bar{t}}, \mathbf{0}_\omega, 0, \mathbf{0}_\omega, [\![s_t]\!], \mathbf{0}_\omega].$$

Thus,

$$\boldsymbol{a}^{(2)}_t = \mathrm{Att}(\boldsymbol{p}^{(2)}_t, \boldsymbol{K}^e_{\bar{t}}, \boldsymbol{V}^e_{\bar{t}}) + \boldsymbol{p}^{(2)}_t = [\boldsymbol{h}_{t-1}, \mathbf{0}_h, \boldsymbol{s}_{\bar{t}}, \boldsymbol{\delta}_{\bar{t}}, \frac{1}{2^{t+1}}, \mathbf{0}_\omega, [\![s_t]\!], \boldsymbol{\omega}_{\bar{t}}].$$

In the final block of the decoder in the second layer, the computation for RNN takes place. In Lemma D.4 below we construct the feed-forward network $O^{(2)}(\cdot)$ such that

$$O^{(2)}(\boldsymbol{a}^{(2)}_t) = [\sigma(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{s}_{\bar{t}} + \boldsymbol{b}) - \boldsymbol{h}_{t-1} \quad \mathbf{0}_h, -\boldsymbol{s}_{\bar{t}}, -\boldsymbol{\delta}_t, 0, \mathbf{0}_\omega, -[\![s_t]\!], \mathbf{0}_\omega]$$

and hence

$$\boldsymbol{z}^{(2)}_t = O^{(2)}(\boldsymbol{a}^{(2)}_t) + \boldsymbol{a}^{(2)}_t = [\sigma(\mathbf{W}_h \boldsymbol{h}_{t-1} + \mathbf{W}_x \boldsymbol{s}_{\bar{t}} + \boldsymbol{b}), \mathbf{0}_h, \mathbf{0}_s, \mathbf{0}_\omega, \frac{1}{2^{t+1}}, \mathbf{0}_\omega, \mathbf{0}_\omega, \boldsymbol{\omega}_{\bar{t}}],$$

which gives

$$\boldsymbol{y}_{t+1} = [\ \boldsymbol{h}_t, \mathbf{0}_h, \mathbf{0}_s, \mathbf{0}_\omega, \tfrac{1}{2^{t+1}}, \mathbf{0}_\omega, \mathbf{0}_\omega, \boldsymbol{\omega}_{\bar{t}}\ ],$$

proving the induction hypothesis (14) for $t+1$, and completing the simulation of RNN. ☐

D.1 Technical Lemmas

Lemma D.2. *There exists a function $O^{(1)}(.)$ defined by feed-forward network such that,*

$$O^{(1)}(\boldsymbol{a}^{(1)}_t) = [\mathbf{0}_h, \mathbf{0}_h, -\boldsymbol{s}_{0:t}, \boldsymbol{\delta}_t, -\frac{1}{2^t} + \frac{1}{2^{t+1}}, \mathbf{0}_\omega, -\boldsymbol{\omega}_t, -\boldsymbol{\omega}_{t-1} + \boldsymbol{\omega}_t]$$

Proof. We define the feed-forward network $O^{(1)}(.)$ such that

$$O^{(1)}(\boldsymbol{a}^{(1)}_t) = [\mathbf{0}_h, \mathbf{0}_h, -\boldsymbol{s}_{0:t}, \boldsymbol{\delta}_t - \boldsymbol{\omega}_t, -\frac{1}{2^t} + \frac{1}{2^{t+1}}, \mathbf{0}_\omega, \mathbf{0}_\omega, -\boldsymbol{\omega}_{t-1} + \boldsymbol{\omega}_t]$$

where

$$\boldsymbol{\delta}_t = (\boldsymbol{\Delta}_{t,1}, \ldots, \boldsymbol{\Delta}_{t,n-1}, 1/2^{t+1}), \quad 0 \le \delta_t \le 1$$

Recall that,

$$\boldsymbol{a}^{(1)}_t = [\ \boldsymbol{h}_{t-1}, \mathbf{0}_h, \boldsymbol{s}_{0:t}, \boldsymbol{\omega}_t, \tfrac{1}{2^t}, \mathbf{0}_\omega, \mathbf{0}_\omega, \boldsymbol{\omega}_{t-1}\ ]$$

We define the feed-forward network $O(\boldsymbol{a}_t)$ as follows,

$$O^{(1)}(\boldsymbol{a}_t) = \mathbf{W}_2 \sigma(\mathbf{W}_1 \boldsymbol{a}^{(1)}_t + \boldsymbol{b}_1)$$

where $\boldsymbol{W}_i \in \mathbb{Q}^{d\times d}$ and $\boldsymbol{b}_1 \in \mathbb{Q}^d$. Define $\boldsymbol{W}_1$ as

$$\begin{array}{c} \\ 2d_h \\ d_e \\ d_\omega - 1 \\ 1 \\ 1 \\ d_\omega \\ d_\omega \\ d_\omega \end{array}\begin{array}{c} \begin{array}{ccccccc} 2d_h & d_e & d_\omega & 1 & d_\omega & d_\omega & d_\omega \end{array} \\ \left[\begin{array}{c|c|c|c|c|c|c} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & -\mathbf{I} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{1}{2} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{1}{2} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{array}\right] \end{array}$$

and $\boldsymbol{b}_1 = \mathbf{0}$, then

$$\sigma(\boldsymbol{W}_1 \boldsymbol{a}_t^{(1)} + \boldsymbol{b}_1) = [\mathbf{0}_h, \mathbf{0}_h, \boldsymbol{s}_{0:t}, \boldsymbol{\Delta}_t, \frac{1}{2^{t+1}}, \boldsymbol{\omega}_t,\ \boldsymbol{\omega}_{t-1},\ \ \boldsymbol{\omega}_{t-1}]$$

We define $\boldsymbol{W}_2$ as

$$\begin{array}{c} \\ 2d_h \\ d_e \\ d_\omega - 1 \\ 1 \\ 1 \\ d_\omega \\ d_\omega \\ d_\omega \end{array}\begin{array}{c} \begin{array}{ccccccc} 2d_h & d_e & d_{\omega-1} & 2 & d_\omega & d_\omega & d_\omega \end{array} \\ \left[\begin{array}{c|c|c|c|c|c|c} \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & -\mathbf{I} & \mathbf{0} & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{I} & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & 1,0 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & -2,1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{I} & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & \mathbf{0} & -\mathbf{I} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & \mathbf{0} & \mathbf{I} & -\mathbf{I} \end{array}\right] \end{array}$$

This leads to

$$O^{(1)}(\boldsymbol{a}_t^{(1)}) = [\mathbf{0}_h, \mathbf{0}_h, \boldsymbol{s}_{0:t}, \boldsymbol{\delta}_t, -\frac{1}{2^t} + \frac{1}{2^{t+1}},\ \mathbf{0}_\omega,\ -\boldsymbol{\omega}_t,\ \ -\boldsymbol{\omega}_{t-1} + \boldsymbol{\omega}_t]$$

which is what we wanted to prove.

□

Lemma D.3. *Let $\boldsymbol{p}_t^{(2)} \in \mathbb{Q}^d$ be a query vector such that*

$$\boldsymbol{p}_t^{(2)} = [\ \cdot,\ \cdot,\ \cdot,\ \boldsymbol{\delta}_t,\ \cdot, \cdot, \cdot, \cdot\ ]$$

where $t \geq 0$ and '$\cdot$' denotes an arbitrary value. Then we have

$$\mathrm{Att}(\boldsymbol{p}_t^{(2)}, \boldsymbol{K}_{\bar{t}}^e, \boldsymbol{V}_{\bar{t}}^e) = [\ \mathbf{0}_h, \mathbf{0}_h, \boldsymbol{s}_{\bar{t}}, \mathbf{0}_\omega, 0, \mathbf{0}_\omega, [\![s_t]\!], \mathbf{0}_\omega\ ]. \tag{16}$$

Proof. Let

$$(\hat{\alpha}_{t,1}^{(2,2)}, \ldots, \hat{\alpha}_{t,\bar{t}}^{(2,2)}) = \mathsf{hardmax}\left(\langle \boldsymbol{p}_t^{(2)}, \boldsymbol{k}_1^e \rangle, \ldots, \langle \boldsymbol{p}_t^{(2)}, \boldsymbol{k}_{\bar{t}}^e \rangle\right)$$

be the vector of normalized attention scores in the decoder-encoder attention block of layer 2 at time t. Then

$$\mathrm{Att}(\boldsymbol{p}_t^{(2)}, \boldsymbol{K}_{\bar{t}}^e, \boldsymbol{V}_{\bar{t}}^e) = \sum_{j=0}^{\bar{t}} \hat{\alpha}_{t,j}^{(2,2)} \boldsymbol{v}_j^e.$$

We claim that

Claim 1. *For $t \geq 0$ we have*

$$(\hat{\alpha}_{t,1}^{(2,2)}, \ldots, \hat{\alpha}_{t,\bar{t}}^{(2,2)}) = \frac{1}{\lambda_{\bar{t}}} \left(\mathbb{I}(s_0 = s_t), \mathbb{I}(s_1 = s_t), \ldots, \mathbb{I}(s_{\bar{t}} = s_t) \right),$$

where λ_t is a normalization factor given by $\lambda_t = \sum_{j=0}^{n-1} \mathbb{I}(s_j = s_t)$.

We now prove the lemma assuming the claim above. Denote the L.H.S. in (16) by $\boldsymbol{\gamma}_t$. Note that if $s_j = s_t$, then $\boldsymbol{v}_j^e = \boldsymbol{\gamma}_t$. Now we have

$$\begin{aligned} \sum_{j=0}^{\bar{t}} \hat{\alpha}_{t,j}^{(2,2)} \boldsymbol{v}_j^e &= \frac{1}{\lambda_t} \sum_{j=0}^{\bar{t}} \mathbb{I}(s_j = s_t)\, \boldsymbol{v}_j^e \\ &= \frac{1}{\lambda_t} \left(\sum_{j=0}^{\bar{t}} \mathbb{I}(s_j = s_t) \right) \boldsymbol{\gamma}_t \\ &= \boldsymbol{\gamma}_t, \end{aligned}$$

completing the proof of the lemma modulo the proof of the claim, which we prove next. □

Proof. (of Claim 1) For $0 < t \leq n$, the vector $\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1}$ has the form

$$\left(\left(\frac{1}{t+1} - \frac{1}{t} \right), \ldots, \left(\frac{\phi_{t,k}}{t+1} - \frac{\phi_{t-1,k}}{t} \right), \ldots, 0 \right).$$

If $s_t = \beta_k$, then

$$\begin{aligned} (\boldsymbol{\omega}_t &- \boldsymbol{\omega}_{t-1})_k && (17) \\ &= \left(\frac{\phi_{t,k}}{t+1} - \frac{\phi_{t-1,k}}{t} \right) && (18) \\ &= \left(\frac{\phi_{t-1,k} + 1}{t+1} - \frac{\phi_{t-1,k}}{t} \right) && (19) \\ &= \frac{t - \phi_{t-1,k}}{t(t+1)} && (20) \\ &\geq \frac{1}{t(t+1)}. && (21) \end{aligned}$$

The last inequality used our assumption that $s_0 = \#$ and that $\#$ does not occur at any later time and

therefore $\phi_{t-1,j} < t$. On the other hand, if $s_t \neq \beta_k$, then

$$\begin{aligned}(\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1})_k &= \left(\frac{\phi_{t,k}}{t+1} - \frac{\phi_{t-1,k}}{t}\right) \\ &= \left(\frac{\phi_{t-1,k}}{t+1} - \frac{\phi_{t-1,k}}{t}\right) \\ &= -\frac{\phi_{t-1,j}}{t(t+1)} \qquad (22) \\ &\leq 0.\end{aligned}$$

This leads to,

$$\begin{aligned}(\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1})_k &> 0 \qquad \text{if } s_t = \beta_k, \\ (\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1})_k &\leq 0 \qquad \text{otherwise.}\end{aligned}$$

In words, the change in the proportion of a symbol is positive from step $t-1$ to t if and only if it is the input symbol at the t-th step. For $0 \leq t \leq n$ and $1 \leq k \leq |\Sigma|$, this leads to

$$\begin{aligned}\boldsymbol{\Delta}_{t,k} = \sigma(\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1})_k &> 0 \qquad \text{if } s_t = \beta_k, \\ \boldsymbol{\Delta}_{t,k} = \sigma(\boldsymbol{\omega}_t - \boldsymbol{\omega}_{t-1})_k &= 0 \qquad \text{otherwise,}\end{aligned}$$

For $t > n$,

$$\boldsymbol{\Delta}_t = \mathbf{0}.$$

Recall that $\boldsymbol{p}_t^{(2)} = \boldsymbol{z}_t^{(1)}$ which comes from (15), and $\boldsymbol{k}_j^e$ is defined in (13). We reproduce these for convenience:

$$\begin{aligned}\boldsymbol{p}_t^{(2)} &= [\ \boldsymbol{h}_{t-1}, \ \mathbf{0}_h, \ \mathbf{0}_s, \\ &\qquad \boldsymbol{\delta}_{\bar{t}}, \ \tfrac{1}{2^{t+1}}, \mathbf{0}_\omega, \mathbf{0}_\omega, \ \boldsymbol{\omega}_{\bar{t}} \ ], \\ \boldsymbol{k}_j^e &= [\ \mathbf{0}_h, \mathbf{0}_h, \mathbf{0}_s, \\ &\qquad [\![s_j]\!], 0, \mathbf{0}_\omega, \mathbf{0}_\omega, \mathbf{0}_\omega \ ].\end{aligned}$$

It now follows that for $0 < t < n$, if $0 \leq j \leq t$ is such that $s_j \neq s_t$, then

$$\langle \boldsymbol{p}_t^{(2)}, \boldsymbol{k}_j^e \rangle = \langle \boldsymbol{\delta}_t, [\![s_j]\!] \rangle = \boldsymbol{\delta}_{t,i} = 0.$$

And for $0 < t < n$, if $0 \leq j \leq t$ is such that $s_j = s_t = \beta_i$, then

$$\begin{aligned}\langle \boldsymbol{p}_t^{(2)}, \boldsymbol{k}_j^e \rangle &= \langle \boldsymbol{\delta}_t, [\![s_j]\!] \rangle = \boldsymbol{\delta}_{t,i} \qquad (23) \\ &= \frac{t - \phi_{t-1,j}}{t(t+1)} \geq \frac{1}{t(t+1)}. \qquad (24)\end{aligned}$$

Thus, for $0 \leq t < n$, in the vector $\left(\langle \boldsymbol{p}_t^{(2)}, \boldsymbol{k}_0^e \rangle, \ldots, \langle \boldsymbol{p}_t^{(2)}, \boldsymbol{k}_t^e \rangle\right)$, the largest coordinates are the ones indexed by j with $s_j = s_t$ and they all equal $\frac{t-\phi_{t-1,i}}{t(t+1)}$. All other coordinates are 0. For $t \geq n$, only the last coordinate $\langle \boldsymbol{p}_t^{(2)}, \boldsymbol{k}_n^e \rangle = \langle \boldsymbol{\delta}_t, [\![\$]\!] \rangle = \frac{1}{2^{t+1}}$ is non-zero. Now the claim follows immediately by the definition of hardmax. □

Lemma D.4. *There exists a function $O^{(2)}(.)$ defined by feed-forward network such that, for $t \geq 0$,*

$$\begin{aligned}O^{(2)}(\boldsymbol{a}_t^{(2)}) = [&\sigma(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_x \boldsymbol{s}_{\bar{t}} + \boldsymbol{b}) - \boldsymbol{h}_{t-1}, \\ &\mathbf{0}_h, -\boldsymbol{s}_{\bar{t}}, -\boldsymbol{\delta}_t, 0, \mathbf{0}_\omega, -[\![s_t]\!], \mathbf{0}_\omega]\end{aligned}$$

where $\boldsymbol{W}_h, \boldsymbol{W}_x$ and $\boldsymbol{b}$ denote the parameters of the RNN under consideration.

Proof. Proof is very similar to proof of lemma C.4. □

E Details of Experiments

In this section, we describe the specifics of our experimental setup. This includes details about the dataset, models, setup and some sample outputs.

E.1 Impact of Residual Connections

The models under consideration are the vanilla Transformer, the one without decoder-encoder residual connection and the one without decoder-decoder residual connection. For the synthetic tasks, we implement a single layer encoder-decoder network with only a single attention head in each block. Our implementation of the Transformer is adapted from the implementation of (Rush, 2018). Table 4 provides some illustrative sample outputs of the models for the copy task.

SOURCE & REFERENCE	– there was no problem at all says douglas ford chief executive officer of the futures exchange
DIRECTIONAL TRANSFORMER	– there was no problem at all says douglas ford chief executive officer of the futures exchange
VANILLA TRANSFORMER	– there was no problem at all says douglas ford chief executive officer

Table 4: Sample outputs by the models on the copy task on length 16. With absolute positional encodings the model overfits on terminal symbol at position 13 and generates sequence of length 12.

For the machine translation task, we use OpenNMT (Klein et al., 2017) for our implementation. For preprocessing the German-English dataset we used the script from fairseq. The dataset contains about 153k training sentences, 7k development sentences and 7k test sentences. The hyperparameters to train the vanilla Transformer were obtained from fairseq's guidelines. We tuned the parameters on the validation set for the two baseline model. To preprocess the English-Vietnamese dataset, we follow Luong and Manning (2015). The dataset contains about 133k training sentences. We use

the tst2012 dataset containing 1.5k sentences for validation and tst2013 containing 1.3k sentences as test set. We use noam optimizer in all our experiments. While tuning the network, we vary the number of layer from 1 to 4, the learning rate, the number of heads, the warmup steps, embedding size and feedforward embedding size.

E.2 Masking and Encodings

Our implementation for directional transformer is based on (Yang et al., 2019) but we use only unidirectional masking as opposed to bidirectional used in their setup. While tuning the models, we vary the layers from 1 to 4, the learning rate, warmup steps and the number of heads.