Turing Completeness of Neural Networks

D'Ambrosi Denis
dambrosi.denis@spes.uniud.it
147681

February 27, 2023

Abstract

Neural networks are a powerful machine learning technique that can be applied to a variety of applications, from image classification to natural language processing. At a basic level, neural networks consist of interconnected nodes that transmit and locally process data, but this seamingly simple architecture allows for an impressive degree of adaptability. Since nowadays they are broadly used to solve virtually any form of task, we must question they actual expressive power: are they actually Turing complete? In this paper, we will examine neural networks' Turing completeness and how it affects how they can be used for computation.

Contents

1 Introduction

> *A neural network is the second best way to solve any problem. The best way is to actually understand the problem.*
>
> *Unknown*

In 1989, Cybenko [Cyb89] showed that for each continuous function there exists at least one neural network with a single hidden layer capable of approximating it to arbitrary accuracy. This result is significant because it implies that neural networks can be used to model a wide range of complex functions, including those with non-linear relationships between input and output variables. Multiple variations to the standard architecture have been proposed and implemented (for example increasing the number of layers, including skip connections and adding loops to the computational graph) during the years, but we must assess whenever these alternatives actually increase the expressive power of the basic topology and if so, where is the boundary of this data structure's computability power. The current essay is structured in the following way: the rest of this section will introduce the foundamental concepts and notation for the rest of the material. In section 2 we'll analyze the Turing completeness of (recurrent) neural networks from a theoretical perspective, without caring about actual implementability of the systems described. In the following section we will instead take a look at concrete architectures that were proposed to simulate memory-bounded Turing machines. Finally, in the conclusions, we'll sum up the results and present some expected future work within this research field.

1.1 Turing Machines

Turing machines are a fundamental concept in the theory of computation, introduced by the mathematician Alan Turing in 1937 [Tur37]. They provide a formal definition of what it means to compute a function, and have been instrumental in advancing our understanding of the limits of what can be computed by a mechanical process.

A Turing machine consists of a tape divided into discrete cells, a read/write head that can move along the tape, and a set of rules that govern how the head interacts with the tape. The tape is initially populated with a finite sequence of symbols, and the machine is in a particular (starting) state. At each step, the machine reads the symbol under the head, performs a specified action (such as writing a new symbol, moving the head left or right, or changing its state), and then moves to the next cell on the tape. The output of the machine is determined by the final state and the symbols on the tape.

Formally, a Turing machine can be represented using a quadruple

$$(Q \cup \{q_{\text{accept}}, q_{\text{reject}}\}, \Gamma, \delta, q_0) \tag{1}$$

where:

- Q is a finite set of states.
- Γ is a finite set of tape symbols, which includes the blank symbol #.
- δ is a transition function that maps $Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, where L and R represent moving the head left or right on the tape.
- $q_0 \in Q$ is the initial state.
- $q_{\text{accept}} \in Q$ is the accepting state.
- $q_{\text{reject}} \in Q$ is the rejecting state.

Before continuing, it's best that we also define the concept of istantaneous description of a Turing machine, since it will represent the starting (and ending) point of a simulation cycle by the neural networks that will be constructed.

An instantaneous description of a Turing machine is a snapshot of its current state. It provides a complete description of the machine's configuration at a given point in time, including the contents of the tape, the position of the head, and the current state of the machine.

Formally, an instantaneous description can be represented as a 3-tuple (q, l, r), where:

- q is the current state of the machine.
- l is the contents of the tape to the left of the head.
- r is the contents of the tape to the right of the head.

For example, if a Turing machine is in state q_0, with the tape contents "00101" and the read/write head positioned over the first symbol, the instantaneous description can be represented as $(q_0, \epsilon, 00101)$, where ϵ represents the empty string.

The instantaneous description of a Turing machine is important because it allows us to reason about its behavior at a particular point in time. By examining the current state and tape contents, we can determine which transition the machine will take next, and how it will update its configuration. This allows us to analyze the computation of the machine step by step, and to understand how it processes input and produces output.

1.2 Neural Networks

Neural networks are a class of machine learning models that are designed to learn and recognize patterns in data. At their core, neural networks are composed of a large number of interconnected processing nodes, which are designed to simulate the behavior of neurons in the brain. As computer scientists, we are able to execute sophisticated computations on input data by connecting these nodes into intricate, layered structures, enabling the data structures themselves

to deduce intricate patterns in the information provides and forecast future predictions on unseen examples.

At the most basic level, a node in a neural network is a mathematical function that takes one or more inputs and produces a single scalar output. Each node is connected to one or more other nodes, composing a network of interconnected processing elements. The computed output of one node serves as the input to the next (that may be just one or multiple ones), allowing information to flow in a predetermined way through the network itself and be processed at each step.

Following the definition used by David Kriesel [Kri07], we can define a neuron as a node that computes the composition of three separate functions:

1. A **net** function, that aggregates the input values from the input neurons
2. An **activation** function, that maps the net value and a threshold value called **bias** into a single scalar value
3. An **output** function, that transforms the "activated value" of the neuron into an output scalar that can be forwarded to the following neurons

In practice, a weighted sum is istantiated as net function, a sigmoid ($\sigma(z) = 1/(1+e^{-z})$) or $ReLU$ ($\sigma(z) = max(0, z)$) function is chosen as activation and the output is left as the identity. Although most of the real architectures do not differ much from this last blueprint, keeping in mind the more general definition will allow us to better investigate the expressive power of these structures.

The weights in the net function are learned through a process called training, which involves adjusting the weights to minimize the error between the network's output and the desired output. This allows the network to learn to recognize patterns in the data and make accurate predictions. In order to do so, we generally use the backpropagation algorithm [quote], that requires all of the operations computed within the network to be differentiable in order to compute a gradient.

It's also important to keep in mind that at least one of the three composed functions (generally the activation) needs to introduce non-linearity to the system: otherwise, the network would be limited to performing linear transformations on the input data, which would severely limit its expressiveness.

Taking a closer look to the types of connection within a network, we can disciminate between two main kinds of architectures:

- **Feedforward Neural Networks** (FFNNs) are networks that have a single flow of input, where the data is processed from the input layer through one or more hidden layers to the output layer. In FFNNs, the information flows in one direction, from the input to the output layer, with no feedback connections: the output of one layer serves as the input for the next layer.
- On the other hand, **Recurrent Neural Networks** (RNNs) are designed to process sequential data, where the order of the data points matters.

RNNs have loops in the network, which allow the output of a given layer to be fed back as input to the same layer or to a previous layer in the network. RNNs can maintain a sort of "memory" of previous inputs, which enables them to handle sequential data such as speech, text, and time series data. The main advantage of RNNs is their ability to model sequences of arbitrary length and process input data of variable size. We will shortly see how this property will be crucial to obtain Turing completeness.

2 Theoretical Turing completeness of RNNs

In this section we will further investigate the expressive power of Recurrent Neural Networks without taking into consideration the actual realizability of the systems taken into consideration. This branch of research has been initially explored by Siegelmann and Sontag in 1995 [SS95], when they presented a theoretical framework for understanding the computational power of neural networks, and in particular, their ability to simulate the behavior of Turing machines.

This result has significant implications for the field of artificial intelligence and computer science, as it shows that neural networks are not just powerful tools for solving specific problems, but can also serve as a universal computational substrate capable of performing any computation that can be performed by a Turing machine.

Unfortunately, this result is not applicable to real-life recurrent architectures since we would require registers to have unbounded precision to store the content of the tape using rational numbers through a fractal encoding function. Still, in 2021 Chung and Siegelmann published another article [CS21] about the expressiveness of recurrent networks: this paper did not introduce any practical implications about the Turing completeness of these data structures but clarified some aspects about the previous proof from 1995 and partially solved the finite precision problem by introducing other caveauts. In the rest of this section we will take a brief look at the results provided by this article, as well as their issues.

2.1 Unbounded precision

The first theorem presented in [CS21] does not add any additional information to the proof from 1995 from a theoretical standpoint, but it provides a clearer explaination and a slighty faster (from 4 to 3 steps to simulate a single Turing machine transition) simulation; it states:

Theorem 1 *Given a Turing Machine $\mathcal{M}$, there exists an injective function $\rho : \mathcal{X} \to \mathbb{Q}^N$ and an n-neuron unbounded-precision RNN $\mathcal{T}_{W,b} : \mathbb{Q}^n \to \mathbb{Q}^n$, where $n = 2|\Gamma| + \lceil \log_2 |Q| \rceil |Q||\Gamma| + 5$ such that for all istantaneous descriptions*

$$\rho^{-1}(\mathcal{T}^3_{W,b}(\rho(x))) = \mathcal{P}_{\mathcal{M}}(x)$$

Let's dissect this statement.

ρ The construction of ρ is provided earlier in the article: it includes a version of the fractal encoding function already introduced in [SS95], that allows us to save an instant configuration into a vector of size $2|\Gamma| + \lceil \log_2 |Q| \rceil |Q||\Gamma| + 5$ of rational numbers. ρ concatenates the encodings of the state, the left tape, the right tape and the first symbols to the sides of the head (even if it's actually explained that these last two informations aren't strictly necessary). We still need to define the encoding functions for the state, the tapes and the symbols, but before we need to note that for this construction to work properly we must encode the symbol alphabet Γ into a set of odd numbers.

- $\rho^{(q)} : Q \to \{0,1\}^{\lceil \log_2 Q \rceil}$ maps the states into a binary enumeration (for example, assuming we have $Q = \{q_0, q_1, q_2\}$, we can define a state encoding as $\rho^{(q)}(q_0) = [0,0],\ \rho^{(q_1)}(q_0) = [0,1],\ \rho^{(q_2)}(q_0) = [1,0]$)
- $\rho^{(s)} : \Gamma* \to \mathbb{Q}$ is used to encode the left and right tape using the following fractal encoding function:

$$\rho^{(s)}(y) := \left(\sum_{i=1}^{|y|} \frac{y_{(i)}}{(2|\Gamma|)^i} \right) + \frac{1}{(2|\Gamma|)^{|y|}(2|\Gamma| - 1)} \tag{2}$$

  Note that using fractal encoding we can manipulate the top symbols of the encoded stack and check for emptyness through simple arithmetical operations as explained in [SS95] and [Sie95].

- $\rho^r : \Gamma \to \{0,1\}^{|\Gamma|-1}$ allows for the encoding of a single symbol $s \in \Gamma$. The formula that defines each of the coordinates is

$$\rho_i^{(r)}(s) = 1\{s > 2i\}\ \forall i \in 1, .., |\Gamma| - 1 \tag{3}$$

  That is, the i-th coordinate of the encoding is equal to 1 if the symbol is bigger then $2i$ and 0 otherwise. A simple example of this encoding would be that if we had $\Gamma = \{1,3,5\}$, then $\rho^{(r)}(1) = [0,0],\ \rho^{(r)}(3) = [1,0],\ \rho^{(r)}(5) = [1,1]$.

References

[CS21] Stephen Chung and Hava T Siegelmann. Turing completeness of bounded-precision recurrent neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[Kri07] David Kriesel. *A brief introduction to neural networks*. BoD–Books on Demand, Norderstedt, Germany, 1 edition, 2007.

[Sie95] Hava T Siegelmann. Computation beyond the turing limit. *Science*, 268(5210):545–548, 1995.

[SS95] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50:132–150, 1995.

[Tur37] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(1):230–265, 1937.