1 Model Framework

We consider continuous functions $f : D \mapsto I$, where $D \subseteq \mathbb{R}^m,\ I \subseteq \mathbb{R}^n$, that are implemented by neural networks. We mostly consider feed-forward networks which implicitly define a mapping from their input neurons to their output neurons.

It might feel strange to analyze feed-forward networks (in contrast to recurrent ones), since, intuitively, the human brain operates recurrently. However, there are multiple reasons for this choice: First, shortcomings of this architectural design help us understand beneficial design patterns. Second, while the entire human brain might operate recurrently, there still might be non-recurrent sub networks. Third, it is a known result from computer science that multilayer perceptrons are universal approximators (if there aren't any parameter constraints).

Theorem 1.1 (MLPs are Universal Approximators)**.** *Given any continuous function $f : D \mapsto I$, where $D \subseteq \mathbb{R}^m,\ I \subseteq \mathbb{R}^n$, and D is compact, there exists a multilayer perceptron that defines a function $f' : D \mapsto I$ s.t.*

$$\max_{\boldsymbol{x} \in D} \|f(\boldsymbol{x}) - f'(\boldsymbol{x})\| < \epsilon$$

for any $\epsilon > 0$.

Clearly, based on the assumption we have to define an encoding of the instances of our problem domain to $\mathbb{R}^n$. This encoding is also invariant in time and space, which must not necessarily be the case in human brains. If you think of the number 1, your brain might show different activity at different times (maybe one time you are more stressed out; or you think differently of the number based on your current activity, as 1 also represents the multiplicative identity for example). Still, there might be some sub-circuitry activated when adding numbers where a certain set of neurons show similar activation patterns when adding by 1.

1.1 Motivation for Symbol Manipulation

Now, there are rather "uninteresting" mappings. To clarify what I mean, let's analyze the following example:

Example 1.1. Let $f : \mathbb{N} \mapsto \{0, 1\}, f(n) \mapsto 1[n \text{ is even}]$. Seemingly, we humans can compute this functions over the entire set $\mathbb{N}$. Note, however, that most digits are of no importance, and hence we don't have to remember and manipulate them. As an illustration, if I asked you whether 985094890282377640950293869 02 is even or odd, you wouldn't have any issues. But if I asked you to repeat this number instead, most of us would fail (even if you managed to do this, what about even bigger and bigger numbers?).

Why did I claim this mapping to be uninteresting? Because this problem can be reduced to a mapping $f' : \{0, 1, \ldots, 9\} \mapsto \{0, 1\}$ (by just considering the last digit). This mapping has a finite domain, and hence also a finite image. Such mappings can be implemented by simple input-output associations (and hence no abstraction or "symbol manipulation" is needed). This leads to the following definition:

Definition 1.1. We say $f : D \mapsto I$ *requires symbol manipulation* iff $f(D)$ is infinite.

Proposition 1.1. *Let $f : D \mapsto I$ with $f(D) = I$, i.e. f is surjective, and let I be infinite. It follows that D is infinite as well.*

Proof. For the sake of contradiction, assume D was finite. But then $f(D)$ must be finite as well. But since $f(D) = I$, I must be finite, a contradiction. $\lightning$ □

Lemma 1.1. *Let $f : D \mapsto I$ be a function with an infinite image $f(D)$. Then f cannot be reduced to another function $g : h(D) \mapsto f(D)$ with finite image satisfying $f(\boldsymbol{x}) = g(h(\boldsymbol{x}))$ for all $\boldsymbol{x} \in D$.*

Proof. Assume that $f(\boldsymbol{x}) = g(h(\boldsymbol{x}))$, i.e. f can be reduced to g. This implies $g(h(D)) = f(D)$, and hence the image of g is infinite. □

Corollary 1.1. *A function f that requires symbol manipulation cannot be reduced to a function that does not require symbol manipulation.*

In the book, the author put much emphasis on so called *universally quantified one-to-one mappings*. Hence, we take over his definition:

Definition 1.2. A function $f : D \mapsto I$ is said to be *UQOTOM* (universally quantified one-to-one mapping) iff f is bijective.

Proposition 1.2. *Let $f : D \mapsto I$ be a UQOTOM, and let D be infinite. Then f requires symbol manipulation.*

Proof. For the sake of contradiction, assume $f(D)$ was finite. But per definition of cardinality, we thus have $|D| = |f(D)|$, and hence D is finite, a contradiction. $\lightning$ □

1.2 Importance of UQOTOMs for Symbol Manipulation

One might ask whether we humans can truly compute mappings with an infinite image:

Disputation 1.1. *Can the human brain truly compute mappings with an infinite image, i.e. produce an infinite amount of outputs given an infinite amount of inputs?*

Let's analyze the very insightful identity mapping $f(\boldsymbol{x}) \coloneqq \boldsymbol{x}$. This could be like repeating a text or a number. Can we truly repeat *any* text of *any* length?

My take would be that it depends on whether we let the brain operate in what I called "in space and over time": If a human sat in an empty room and was shown an entire roman, would he be able to repeat it? I don't think so. Of course, if the roman was provided sequentially to the human, like word for word, the human could repeat the roman word for word. But this is not the modus operandi of a mapping, where we take the full input at once, and produce the desired output.

If, however, the participant was allowed to use pen and paper, he could copy the input onto that using the fact that he has an infinite amount of time and an infinite amount of paper. This, of course, is not very realistic, but the point is that the human brain seems to be able to imitate an universal computer.

Axiom 1.1. *The human brain can emulate an universal computer given enough time and resources.*

This is especially evident when trying to emulate a Turing machine, which has a finite description, and does small computational steps each time.

But the realistic answer seems to be that humans cannot really compute mappings with an infinite image, and hence cannot really compute abstract symbol manipulation over an infinite domain, though they truly can do so for a finite domain.

But what does it mean to do symbol manipulation over a finite domain? Well, the answer is not that obvious, but consider the mapping $f(x) \coloneqq 42 \cdot x$. I am sure that with a little bit of concentration the reader is able to compute, say, $f(9)$. But how did the brain do that? There is a real chance that you encountered this arithmetic task for the first time in your life.

The idea is that the brain employed some kind of *symbol manipulation*. This means that it is not the case that the brain has an associative memory of all (x, y) pairs and the corresponding product $x \cdot y$. Even though it could use associative memory to compute multiplication since we restricted ourselves to a finite domain, like for example that x and y are less than or equal 100, it seems very unlikely because the brain seems to be able to generalize well to never encountered tasks. (Or did you memorize all the $\binom{100}{2} = 4950$ possible multiplications?)

Definition 1.3 (Informal; Dependent on Situation)**.** A model is said to *generalize well* iff it can compute reasonable outputs for never seen inputs (reasonable with respect to a formal system for example).

Remark 1.1. This property to generalize well is just one aspect of symbol manipulation. This concept corresponds loosely to what the author refers to as *relation over variables*.

As it turns out, the human brain can generalize a variety of mappings well, and many of them are UQOTOMs, i.e. one-to-one mappings. Just to name a few:

- adding or multiplying by a constant (other than 0 or 1)
- repeating the input ($\equiv$ identity function)
- duplicating the input
- producing simple past tense of verbs (like think $\mapsto$ thought)
- $x \mapsto x^2,\ x \in \mathbb{N}$
- ...

The premise is that we humans have a bias for such mappings.

Example 1.2. Say you are given the following training data set of input-output pairs:

Input	Output
1010	1010
0100	0100
1110	1110
0000	0000

What would you guess the output should be for 1111? Most would answer 1111, as it fits the identity pattern of the training data. However, note that the last bit of each output is always a 0, and hence $1111 \mapsto 1110$ would also be reasonable (and in fact many other mappings too).

Even though there are other examples where participants would not generalize across the entire input domain, like presented in the book, we still have the following premise:

Premise 1.1 (Bias for UQOTOMs)**.** *Given a training dataset of an UQOTOM, humans have a bias to abstract an universally quantified mapping that is one-to-one (and hence in a sense generalize well).*

Based on these observations, our goal is to analyze different neural network architectures and to examine their generalization ability, and how their bias corresponds to that of humans.

Remark 1.2. It is questionable, though, whether the human brain wires a new small neural network for each learning task, like one presented in example 1.2. It would also be possible that the brain employs a meta prediction model, that given training data and an input, it produces an output. Hence, the training data is rather a variable fed into this meta prediction network, than serving as a training supervisor for a newly learned neural network.

It may also be the case that the brain reduces functions to functions with a finite image by using its computational ability to emulate basic algorithms. Hence, the brain might store this finite description of the algorithm as well as the basic finite mapping in order to emulate generalizable mappings.

Example 1.3. Take multiplication for example. When multiplying two numbers, like $42 \cdot 9$, does the brain really employ a feed-forward structure to compute this, or does it rather decompose this task to

$$42 \cdot 9 = 4 \cdot 9 \cdot 10 + 2 \cdot 9 \quad ?$$

We see that multiplication can be decomposed to addition and shifting based on a finite algorithm, and addition itself behaves similarly. The algorithm for multiplying a number A by a digit B may look something like this (if B was bigger than we would recursively call this function and add the shifted results):

Algorithm 1 Multiplication by Decomposition

Require: Two positive integers A and B, $B \leq 9$

Ensure: The product $P = A \cdot B$

1: Decompose A into base-10 digits: $A = \sum_{i=0}^{n-1} a_i \cdot 10^i$, where a_i is the i-th digit
2: Initialize $P \leftarrow 0$
3: **for** $i = 0$ to $n - 1$ **do**
4: $\quad P \leftarrow P + (a_i \cdot B) \cdot 10^i$
5: **end for**
6: **return** P