Symbol Manipulation in Neural Networks

Jonas Peters, Philipp Haack

July 2, 2025

Contents

1 Introduction

The human brain is a remarkably complex and efficient system, capable of performing an extraordinary range of cognitive tasks with ease. Its ability to learn from experience, adapt to new environments, and process vast amounts of sensory information has long fascinated scientists and engineers alike.

The study of its functionality is partially in the domain of computer science: We try to model the brain with computer algorithms, which is inspired by the idea that the human brain consists of elementary computational units, neurons, which itself can be sufficiently modeled with computers, and whose interplay creates the emergent complex behavior we see.

We structure this work by logical statements like *axiom*, *premise* and *conclusion* etc. similar to mathematical writing. I believe the chain of argument is more evident this way.

Axiom 1.1. *In theory, the human brain can be sufficiently modelled by a computer program.*

The human brain consists of a very large number of neurons, around 86 billion of them. Every neuron is connected to others via *synapses.* On average, 1,000 synapses are emanating per neuron. If we were to store each synapse by one byte of computer memory, we would need 86 trillion bytes $\approx$ 86 terabytes of storage. Sounds manageable, but how long would it take to run this program? Additionally, note that we stated that each neuron itself acts as an elementary computational unit, i.e. all neurons operate in parallel. Clearly, the complexity of modelling the brain adds up enormously.

We also must not forget that the human brain is a dynamic system which operates *in space and over time.* This is in strong contrast to basic neural networks, which operate in simple input $\mapsto$ output fashion.

Example 1.1. Given the task to multiply 96459 and 29537. We humans can use the space dimension to grab ourselves a pen and a piece of paper, and use the time dimension to repeatedly perform simple multiplications and additions in order to solve the original task. A basic neural network is incapable of that.

Thus, we should think of the human brain as a device which can *create* and *perform* instructions. Hence, we may focus on the operations it can perform as a beginning. We can think of this as a human sitting in an empty room without any auxiliary items around performing a task like adding two numbers.

Premise 1.1. *Among other things, the human brain can perform operations like adding and multiplying numbers (for a restricted domain of $\mathbb{Z}$ for example), or inflecting verbs.*

Thus, we should be able to reconstruct this behavior with our models as well. Of course, it might be the case that all the operations the human brain can perform are an emergent phenomena of the entire brain working as a whole. However, it also could be the case that the brain has certain sub-circuitry for specific tasks like adding numbers. Assuming this is the case, we also might conclude that these circuits shouldn't be much more complex than necessary, as the brain is a product of the energy minimization of evolution.

Remark 1.1. However, this begs the question whether there is some efficient circuitry "preinstalled" due to evolution, or whether everything needs to be learned. But are the learned circuits efficient themselves, or are they rather big and all operations the brain is capable of are emergent phenomena from different brain regions working together?

Based on these observations, our main focus is on how models, specifically neural networks, can efficiently implement a certain subset of tasks humans are naturally good at.

2 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 2.1 (MLPs are Universal Approximators). *Given any continuous function $f : D \mapsto I$, where $D \subseteq \mathbb{R}^m,\ I \subseteq \mathbb{R}^n$, we have for any $\epsilon > 0$ that there exists a multilayer perceptron that defines a function $f' : D \mapsto I$ s.t.*

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

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.

2.1 Motivation for Symbol Manipulation

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

Example 2.1. 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 98509489028237764095029386902 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 2.1. We say $f : D \mapsto I$ *requires symbol manipulation* iff $f(D)$ is infinite.

Proposition 2.1. *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. $\lightning$ $\square$

Lemma 2.1. *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. $\square$

Corollary 2.1. *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 2.2. A function $f : D \mapsto I$ is said to be *UQOTOM* (universally quantified one-to-one mapping) iff f is bijective.

Proposition 2.2. *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. $\lightning$ $\square$

2.2 Importance of UQOTOMs for Symbol Manipulation

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

Disputation 2.1. *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 2.1. *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 2.3 (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 2.1. 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
- 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 2.2. 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 2.1 (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 2.2. It is questionable, though, whether the human brain wires a new small neural network for each learning task, like one presented in example 2.2. 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 2.3. 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:     $P \leftarrow P + (a_i \cdot B) \cdot 10^i$
5: end for
6: return $P$
```

3 Implementing Symbol Manipulation in MLPs

When specifying the structure of a neural network, we restrict our hypothesis class, i.e. we restrict the possible mappings from the input to the output space. Furthermore, we also encode a *bias* into the network, i.e. a certain tendency to prefer some mappings over others.

TODO: example interpolation etc.

Our goal is to encode the premised human bias for UQOTOMs (at least for certain tasks) to MLPs in order to achieve good generalization of these models. We hope that by examining successful models we might be able to draw conclusions to the inner mechanisms of the human brain.

Understanding the bias and learning behavior of multilayer perceptrons is rather difficult, which is why empirical testing is used to assess different architectures.

3.1 MLPs with Linear Activation Functions

When restricting our models to only using linear activation function, though, we can infer some information:

Proposition 3.1. *The hypothesis class of a MLP with linear activation functions is restricted to linear functions $f : \mathbb{R}^m \mapsto \mathbb{R}^n$, i.e. there exists a matrix $\boldsymbol{M} \in \mathbb{R}^{n\times m}$ s.t. $f(\boldsymbol{v}) = \boldsymbol{M}\boldsymbol{v}$.*

Proof. Consider a multi-layer perceptron with L layers and linear activation functions $\phi(x) = \alpha x$. Let the input be $\boldsymbol{v} \in \mathbb{R}^m$, and let each layer i have a weight matrix $\boldsymbol{W}^{(i)}$. The output of the network is:

$$f(\boldsymbol{v}) = \phi\left(\boldsymbol{W}^{(L)}\phi\left(\boldsymbol{W}^{(L-1)}\phi\left(\cdots\phi\left(\boldsymbol{W}^{(1)}\boldsymbol{v}\right)\cdots\right)\right)\right) \quad ,$$

where ϕ is applied element-wise. Since $\phi(x) = \alpha x$, this simplifies to:

$$f(\boldsymbol{v}) = \alpha^L \boldsymbol{W}^{(L)} \boldsymbol{W}^{(L-1)} \cdots \boldsymbol{W}^{(1)} \boldsymbol{v} \quad .$$

Let $\boldsymbol{M} = \alpha^L \boldsymbol{W}^{(L)} \boldsymbol{W}^{(L-1)} \cdots \boldsymbol{W}^{(1)} \in \mathbb{R}^{n \times m}$. Then $f(\boldsymbol{v}) = \boldsymbol{M}\boldsymbol{v}$, which is a linear function. Thus, the hypothesis class is restricted to linear mappings from $\mathbb{R}^m$ to $\mathbb{R}^n$. $\square$

Remark 3.1. This proposition also holds when allowing α to vary by layer, or even with every node.

Lemma 3.1. *Let $\boldsymbol{M} \in \mathbb{R}^{n \times m}$ be a matrix inducing the mapping $f(\boldsymbol{v}) := \boldsymbol{M}\boldsymbol{v}$. Then we have*

$$f \textit{ injective} \iff \operatorname{rank} \boldsymbol{M} = m \quad .$$

Proof. The function f is injective iff $\ker(\boldsymbol{M}) = \{\boldsymbol{0}\}$:

' $\implies$ ' is trivial. For ' $\impliedby$ ', consider the contraposition 'f is not injective $\implies \ker(\boldsymbol{M}) \neq \{\boldsymbol{0}\}$'. Since f is not injective, there must be $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m$ with $\boldsymbol{u} \neq \boldsymbol{v}$ s.t. $\boldsymbol{M}\boldsymbol{u} = \boldsymbol{M}\boldsymbol{v}$. Hence, $\boldsymbol{M}(\boldsymbol{u} - \boldsymbol{v}) = \boldsymbol{0}$, and thus $(\boldsymbol{u} - \boldsymbol{v}) \in \ker \boldsymbol{M}$. Note that $(\boldsymbol{u} - \boldsymbol{v}) \neq \boldsymbol{0}$.

Now, by the rank-nullity theorem, we have:

$$\dim(\ker(\boldsymbol{M})) + \operatorname{rank}(\boldsymbol{M}) = m \quad .$$

Finally, we see that

$$\operatorname{rank}(\boldsymbol{M}) = m \iff \dim(\ker(\boldsymbol{M})) = 0 \iff \ker(\boldsymbol{M}) = \{\boldsymbol{0}\} \iff f \text{ injective} \quad .$$

$\square$

Corollary 3.1. *A MLP with only one input node and linear activations is forced to learn either $f : \mathbb{R} \mapsto \mathbb{R}^n, f(x) \equiv \boldsymbol{0}$ or an UQOTOM.*

Proof. Based on proposition 3.1 we know that f can be written as $f(x) = \boldsymbol{v}x$ for some $v \in \mathbb{R}^{n \times 1}$. Furthermore, based on lemma 3.1 we know that f injective $\iff \operatorname{rank} \boldsymbol{v} = 1$. Since $\boldsymbol{v}$ is a vector, we have $\operatorname{rank} \boldsymbol{v} = 1 \iff \boldsymbol{v} \neq \boldsymbol{0}$.

Hence, for $\boldsymbol{v} \neq \boldsymbol{0}$ we have that f is injective. When restricting the image domain accordingly we also have that f is surjective, and hence UQOTOM.

On the other hand, if $\boldsymbol{v} = \boldsymbol{0}$, then $f(x) \equiv \boldsymbol{0}$. $\square$

Remark 3.2. If a MLP with linear activations has multiple input nodes, it will depend on the properties of matrix $\boldsymbol{M}$ whether or not the MLP implements an UQOTOM based on lemma 3.1.

For example, consider the mapping

$$f : \mathbb{R}^2 \mapsto \mathbb{R}^2, \boldsymbol{v} \mapsto \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \boldsymbol{v} \quad .$$

It is not injective, since $f \begin{pmatrix} 1 \\ 0 \end{pmatrix} = f \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Such a mapping can be implemented by the MLP depicted in figure 3.2.

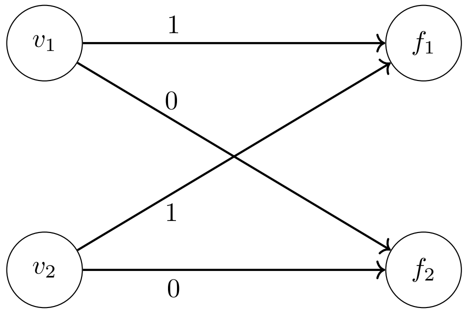

Figure 1: Simple MLP with linear activations implementing a non-injective mapping.

3.2 MLPs with Non-Linear Activation Functions

In order to learn not just linear mappings, MLP typically employ non-linear sigmoidal activation functions, like the logistic function $\sigma(x) := \frac{1}{1+e^{-x}}$ or $\tanh(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

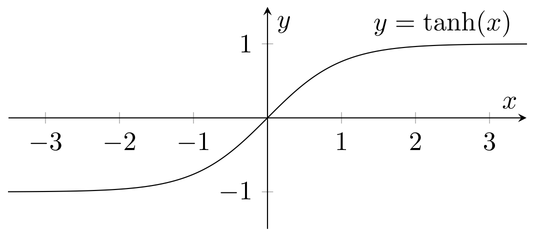

Figure 2: Plot of the function $y = \tanh(x)$.

As it turns out, the analysis of such MLPs with non-linear activations is much harder.

Example 3.1. MLPs with non-linear activations like $\tanh(x)$ can represent non-injective functions (other than $f(x) \equiv \mathbf{0}$) even when only using one input node.

For instance, the MLP depicted in figure 3 implements the non-injective mapping depicted in figure 4.

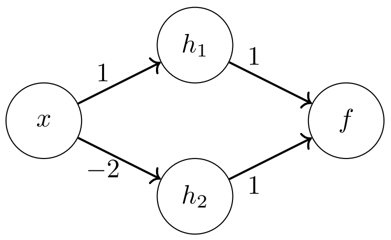

Figure 3: Simple MLP using $\tanh(x)$ activation implementing a non-injective mapping.

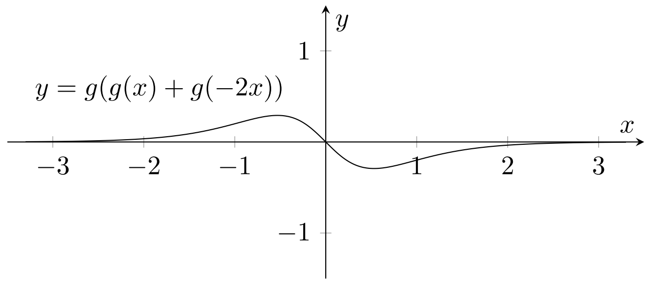

Figure 4: Plot of the function $y = g(g(x) + g(-2x))$ with $g(x) \coloneqq \tanh(x)$.

3.2.1 Multiple Input Nodes

As one might expect, the situation gets even more complicated when considering multiple input nodes. Still, we can observe interesting learning behavior of MLPs and examine how their tendency to abstract functions match to those of humans.

One illustrative example given in the book is to train a MLP with four binary input and output neurons to learn the identity mapping. For example, 1010 is mapped to 1010, 0001 to 0001, and so on. As we can see, this mapping has a finite domain of 16 possible inputs. It would be interesting to analyze how the MLP generalizes when trained on a real subset of the *input space*.

Let's analyze the case of only training the MLP with inputs where the last bit is 0. Formally, our training data set $\mathcal{D}$ reads as:

$$\mathcal{D} = \{(\boldsymbol{x}, \boldsymbol{x}) : \boldsymbol{x} \in \{0,1\}^4,\ \boldsymbol{x}_4 = 0\} \quad .$$

Example 3.2. When you are presented the pattern

Input	Output
1010	1010
0100	0100
1110	1110
0000	0000

and asked to predict the output of 1111, many would predict 1111. Note that there is no right or wrong, as potentially any of the 16 possible bit strings are valid. However, we can infer some information about the human bias, which we can use to examine our models.

Similarly, after training is completed, we can test what the networks prediction would be for inputs like 1111, which it has never seen before. Interestingly, it predicts 1110. I mean, can we blame it, as it never learned how to handle the last bit. But that is exactly the point: The network could alternatively generalize the behavior it learned for the other three nodes to the last one, but instead it treats them independently. This, essentially, represents a bias of the architecture, which for some tasks (like this one) might be unwanted.

3.3 Conclusion

We suspect that brains operate fundamentally differently from MLPs. Unlike feedforward networks, brains function recurrently and learn dynamically, without needing to construct distinct neural architectures for every task. However, the expressive power of MLPs and similar feedforward models allows them to emulate specific tasks at which humans excel—such as language processing and image recognition. When trained on a subset of the input domain, these models often perform poorly at generalization.

The argument is that this misalignment in inductive bias between humans and such models stems from their architectural design. Neural networks are abstract mathematical frameworks that process numbers, with no built-in prior knowledge of the world or the task. While engineers do attempt to incorporate domain-specific knowledge—such as using convolutional layers to recognize shift-invariant features in 2D images—these additions remain narrow in scope.

Personally, I don't believe we should model the brain by designing individual neural networks for specific tasks, as this would require tailoring an architecture for each task that aligns well with human biases. Instead, we should aim to build a single universal model—possibly composed of modular subcomponents—analogous to how a computer includes a CPU, GPU, RAM, and other parts. Such a model should possess basic computational abilities and be capable of coordinating and utilizing its different components to handle a wide range of tasks. In fact, the diversity of tasks this model can perform may enhance its precision, as it could integrate knowledge across domains.

Of course, developing such a system is monumentally difficult. But the potential of this form of *artificial general intelligence* is vast: we can scale and deploy these systems continuously, potentially allowing them to operate more efficiently than humans. The central challenge lies in discovering an appropriate architecture—while always keeping in mind the fundamental differences between the human brain and digital computers, such as the massively parallel nature of biological neural processing.