1 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.

1.1 MLPs with Linear Activation Functions

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

Proposition 1.1. *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 $\mathbb{R}^n$. □

Remark 1.1. This proposition also holds when allowing α to vary by layer, or even with every node.

Lemma 1.1. *Let $\boldsymbol{M} \in \mathbb{R}^{n \times m}$ be a matrix inducing the mapping $f(\boldsymbol{v}) \coloneqq \boldsymbol{M}\boldsymbol{v}$. Then we have*

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

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

' $\Longrightarrow$ ' is trivial. For ' $\Longleftarrow$ ', consider the contraposition 'f is not injective $\Longrightarrow$ $\ker(\boldsymbol{M}) \neq \{\mathbf{0}\}$'. 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, $\boldsymbol{M}(\boldsymbol{u}-\boldsymbol{v}) = \mathbf{0}$, and thus $(\boldsymbol{u}-\boldsymbol{v}) \in \ker \boldsymbol{M}$. Note that $(\boldsymbol{u}-\boldsymbol{v}) \neq \mathbf{0}$.

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

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

Finally, we see that

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

□

Corollary 1.1. *A MLP with only one input node and linear activations is forced to learn either $f : \mathbb{R} \mapsto \mathbb{R}^n, f(x) \equiv \mathbf{0}$ or an UQOTOM.*

Proof. Based on proposition 1.1 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 1.1 we know that f injective $\iff$ $\operatorname{rank} \boldsymbol{v} = 1$. Since $\boldsymbol{v}$ is a vector, we have $\operatorname{rank} \boldsymbol{v} = 1 \iff \boldsymbol{v} \neq \mathbf{0}$.

Hence, for $\boldsymbol{v} \neq \mathbf{0}$ 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 $\boldsymbol{v} = \mathbf{0}$, then $f(x) \equiv \mathbf{0}$. □

Remark 1.2. 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 1.1.

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 1.2.

1.2 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 $\sigma(x) \coloneqq \frac{1}{1+e^{-x}}$ or $\tanh(x) \coloneqq \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

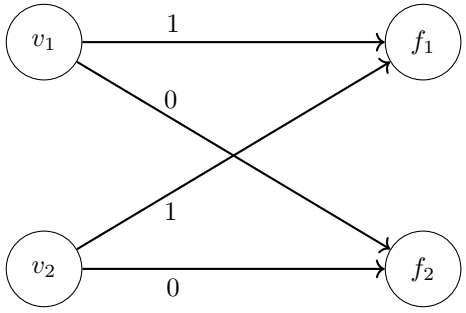

Figure 1: Simple MLP with linear activations implementing a non-injective mapping.

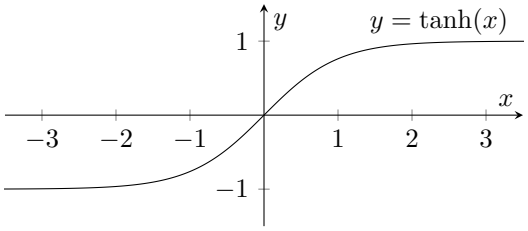

Figure 2: Plot of the function $y = \tanh(x)$.

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

Example 1.1. 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.

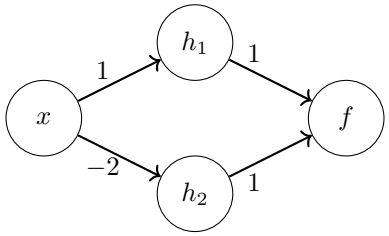

Figure 3: Simple MLP using $\tanh(x)$ activation implementing a non-injective mapping.

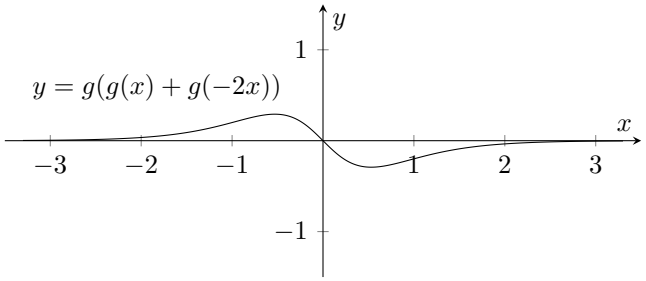

Figure 4: Plot of the function $y = g(g(x) + g(-2x))$ with $g(x) \coloneqq \tanh(x)$.

1.2.1 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 1.2. 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.

1.3 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.