1 Relations between Variables

Marcus starts with the following observation:

Observation 1.0.1. The human brain is capable of performing operations like addition, multiplication, inflecting verbs, and so on.

What do all these algebraic operations have in common? They all operate on variables based on rules. Thus, Marcus calls them *relations between variables.*

1.1 UQOTOM

Instead of considering all these functions of human cognition at once, Marcus focusses on a very special subclass of operations. He refers to them as *UQOTOM*, which stands for *universally quantified one-to-one mapping.* Like the name suggests, they are defined as follows:

Definition 1.1.1 (UQOTOM)**.** A mapping $f : X \to Y$ is a UQOTOM if it is *one-to-one* and *universally quantified.* Universally quantified specifies that f is defined for all $x \in X$, i.e. that f is a valid function in the mathematical sense.

Some important examples of UQOTOMs that Marcus is going to analyze are the identity function, and the simple past operation with produces the simple past of a verb. According to Marcus:

Marcus (2001)

I do not mean to suggest that UQOTOM are the only mappings people compute. But UQOTOM are especially important to the arguments that follow because they are functions in which every new input has a new output. Because free generalization of UQOTOM would preclude memorization, evidence that people (or other organisms) can freely generalize UQOTOM would be particularly strong evidence in support of the thesis that people (or other organisms) can perform operations over variables.

Based on this quote, we can see that Marcus is interested in UQOTOMs because they enforce a system to *generalize* mappings. That is, given a never-before-seen input, a potential model implementing this UQOTOM must produce a new output that has not been produced before. And this generalization is a central aspect of *symbol manipulation* according to Marcus.

Marcus (2001)

To a system that can make use of algebralike operations over variables, free generalization comes naturally.

Furthermore, he specifies algebraic rules further and he writes:

Marcus (2001)

Algebraic rules are not finite tables of memorized facts or relationships between specific instances but open-ended relationships that can be freely generalized to all elements within some class.

Most decisively, for Marcus, a UQOTOM isn't just a special type of function; it is the very implementation of an algebraic rule. He writes:

Marcus (2001)

When such a network represents identity or some other UQOTOM, it represents an abstract relationship between variables—which is to say that such a network implements an algebraic rule.

Critique

The implication

$$f \text{ is a UQOTOM} \implies f \text{ implements an algebraic rule}$$

is too simple and doesn't grasp the complexity of algebraic rules. It is also not a suitable indicator for whether a model implements algebraic rules. For example, the function $f(x) = x^2$ is not a UQOTOM, but it is an algebraic rule. The fundamental flaw is that one cannot reduce the analysis of algebraic rules to the analysis of UQOTOMs.

1.1.1 Humans can freely generalize UQOTOMs

Now that Marcus has established his framework of UQOTOMs, he argues that humans can freely generalize these UQOTOMs. He provides evidence for this claim with the following example:

Example 1.1.1. When you are presented the pattern and asked to predict the output

Input	Output
1010	1010
0100	0100
1110	1110
0000	0000
1111	?

of 1111, what would you answer?

Marcus argues that most people would answer 1111, because humans have a bias for generalizing UQOTOMs, which is why the identity mapping is generalized in this example. He concludes:

Premise 1.1.1. *Humans can freely generalize UQOTOMs.*

1.1.2 Multilayer Perceptrons and Operations over Variables

Next, Marcus emphasizes the distinction between models that allocate *one node per variable*, and models which allocate *multiple nodes per variable*:

Definition 1.1.2. A model is said to allocate *one node per variable* if it has a single node for each variable in the input. A model is said to allocate *multiple nodes per variable* if it has multiple nodes for each variable in the input.

Marcus (2001)

Again, what is relevant here is not the sheer number of input units but rather the number of input units allocated to representing each input variable.

Critique

In the mathematical setting of neural networks that represent one-to-one mappings, it makes no difference whether, say, 5 input nodes all together represent a single variable, or five individual variables all encoded by a single node each. Furthermore, he also does not apply this framework, or only very vaguely, as we will see in the next section.

Additionally, he points out that this distinction is not to be confused with *localist* and *distributed* representations:

Marcus (2001)

While all models that use distributed representations allocate more than one variable per node, it is not the case that all localist models allocate a single node per variable.

With this definition established, Marcus claims that:

Marcus (2001)

One-node-per-variable models, it turns out, can and indeed (the caveats in note 5 notwithstanding) must represent universally quantified one-to-one mappings.

Critique

Again, he assumes the network to only have one input node, which is not the same as having one node per variable. Furthermore, in note 5 he explains that he assumes the network to only have linear activation functions. Such a network must learn a linear mapping, and every mapping $f(x) = \alpha \cdot x$ is one-to-one for $\alpha \neq 0$.
Note that if we had multiple input nodes instead, a linear mapping is described by a matrix multiplication $f(\boldsymbol{x}) = A \cdot \boldsymbol{x}$. In this case, the mapping is one-to-one if and only if A is invertible, which is not guaranteed for all matrices.

Critique

This described case of a single input node with linear activation functions is very specific and doesn't serve any general argument. Note that when using non-linear activation functions, one can easily construct a network representing a non one-to-one mapping:

Example 1.1.2. 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 1.1 implements the non-injective mapping depicted in figure 1.2.

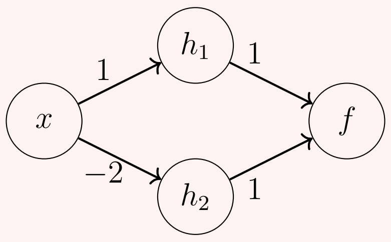

Figure 1.1: Simple MLP using $\tanh(x)$ activation implementing a non-injective mapping.

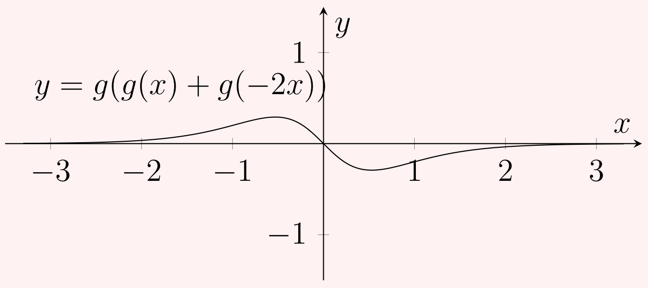

Figure 1.2: Plot of the function $y = g(g(x) + g(-2x))$ with $g(x) \coloneqq \tanh(x)$.

Regarding models that allocate more than one node per variable, Marcus states:

Marcus (2001)

Models that allocate more than one node per variable too, can represent universally quantified one-to-one mappings (see, for example, the left panel of figure 3.4), but they do not have to (see the right panel of figure 3.4).

Critique

This statement caries very little information. It only states that there exists MLPs that represent UQOTOMs, and that there exist MLPs that do not represent UQOTOMs, which is trivially true. The bigger problem, however, is that he also does not provide any further details on how to construct such MLPs, or what the implications of this distinction are.

Learning

Until now, Marcus has only discussed the ability of MLPs to represent UQOTOMs. Now, he turns to the question of whether MLPs can *learn* UQOTOMs. He states:

Marcus (2001)

The flexibility in what multiple-nodes-per-variable models can represent leads to a flexibility in what they can learn. Multiple- nodes-per-variable models can learn UQOTOMs, and they can learn arbitrary mappings. But what they learn depends on the nature of the learning algorithm. Back-propagation—the learning algorithm most commonly used—does not allocate special status to UQOTOMs. Instead, a many-nodes-per-variable multilayer perceptron that is trained by back-propagation can learn a UQOTOM—such as identity, multiplication, or concatenation—only if it sees that UQOTOM illustrated with respect to each possible input and output node.

He justifies that back-propagation does not allocate special status to UQOTOMs with an exapmle:

Example 1.1.3. The autoencoder network depicted in 1.3 is trained to learn the identity mapping:

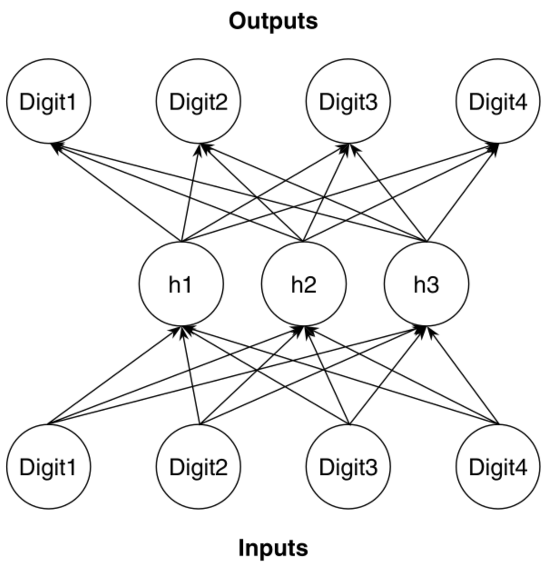

Figure 1.3: Autoencoder network trained to learn the identity mapping.

However, all training samples have a 0 as a last bit. Marcus' simulations show that the network learns to ignore the last bit by always predicting a 0, and only learns the identity mapping for the first three bits.

He formalizes this failure of back-propagation to learn UQOTOMs by defining the concept of *training independence*:

Marcus (2001)

The equations lead to two properties that I call input independence and output independence or, collectively, training independence (Marcus, 1998c).

Definition 1.1.3 (Input Independence)**.** "*Input independence is about how the connections that emanate from input nodes are trained. First, when an input node is always off (that is, set to 0), the connections that emanate from it will never change. This is because the term in the equation that determines the size of the weight change for a given connection from input node x into the rest of the network is always multiplied by the activation of input node x; if the activation of input node x is 0, the connection weight does not change.*" (Marcus, 2001, p. 47)

Definition 1.1.4 (Output Independence)**.** "*Output independence is about the connections that feed into the out- put units. The equations that adjust the weights feeding an output unit j depend on the difference between the observed output for unit j and the target output for unit j but not on the observed values or target values of any other unit. Thus the way the network adjusts the weights feeding out- put node j must be independent of the way the network adjusts the weights feeding output node k (assuming that nodes j and k are distinct).*" (Marcus, 2001, p. 47)