Learning Functors using Gradient Descent

Bruno Gavranović

University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia

Neural networks have become an increasingly popular tool for solving many real-world problems. They are a general framework for differentiable optimization which includes many other machine learning approaches as special cases. In this paper we build a categorical formalism around a class of neural networks exemplified by CycleGAN [13]. CycleGAN is a *collection of neural networks*, closed under composition, whose inductive bias is increased by enforcing composition invariants, i.e. cycle-consistencies. Inspired by Functorial Data Migration [12], we specify the interconnection of these networks using a *categorical schema*, and network instances as set-valued functors on this schema. We also frame neural network architectures, datasets, models, and a number of other concepts in a categorical setting and thus show a special class of *functors*, rather than functions, can be learned using gradient descent. We use the category-theoretic framework to conceive a novel neural network architecture whose goal is to learn the task of object insertion and object deletion in images with unpaired data. We test the architecture on a CelebA dataset and obtain promising results.

1 Introduction

Compositionality describes and quantifies how complex things can be assembled out of simpler parts. It is a principle which tells us that the design of abstractions in a system needs to be done in such a way that we can intentionally forget their internal structure [7]. There are two interesting properties of neural networks related to compositionality: (i) they *are* compositional – increasing the number of layers tends to yield better performance, and (ii) they are discovering (compositional) structures in data.

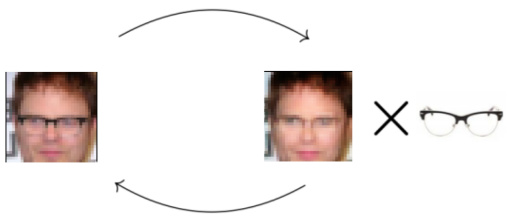


Figure 1: We devise a novel method to regularize neural network training using a category presented by generators and relations. We train neural networks to **remove glasses** from the face of a person and **insert them** parametrically, without ever telling the neural networks that the image contains glasses or even that it contains a person.

Furthermore, an increasing number of components of a modern deep learning system is learned. For instance, Generative Adversarial Networks [6] learn the *cost function*. The paper *Learning to Learn by gradient descent by gradient descent* [2] specifies networks that learn the *optimization function*. The paper *Decoupled Neural Interfaces using Synthetic Gradients* [8] specifies how gradients themselves can be learned. The system in [8] can be thought of as a cooperative multi-player game, where some players depend on other ones to learn but can be trained in an asynchronous manner.

These are just rough examples, but they give a sense of things to come. As more and more components of these systems stop being fixed throughout training, there is an increasingly larger need for more precise formal specification of the things that *do* stay fixed. This is not an easy task; the invariants across all these networks seem to be rather abstract and hard to describe. In this paper we explore the hypothesis that the language of category theory could be well suited to describe these systems in a precise manner.

Bruno Gavranović: bruno@brunogavranovic.com

Inter-domain mappings. Recent advances in neural networks describe and quantify the process of discovering high-level, abstract structure in data using gradient information. As such, learning inter-domain mappings has received increasing attention in recent years, especially in the context of *unpaired data* and image-to-image translation [1, 13]. *Pairedness* of datasets X and Y generally refers to the existence of some invertible function $X \to Y$. Note that in classification we might also refer to the input dataset as being paired with the dataset of labels, although the meaning is slightly different as we cannot obviously invert a label $f(x)$ for some $x \in X$.

Obtaining datasets that contain extra information about inter-domain relationships is a much more difficult task than just the collection of the relevant datasets. Consider the task of object removal from images; obtaining pairs of images where one of them lacks a certain object, with everything else the same, is much more difficult than the mere task of obtaining two sets of images: one that contains that object and one that does not, with everything else varying. Moreover, we further motivate this example by the reminiscence of the way humans reason about the missing object: simply by observing two unpaired sets of images, where we are told one set of images lack an object, we are able to learn what the missing object looks like.

Motivated by the sucess of Generative Adversarial Networks (GANs) [6] in image generation, some existing unsupervised learning methods [1, 13] use adversarial losses to learn the true data distribution of given domains of natural images and *cycle-consistency* losses to learn *coherent* mappings between those domains. CycleGAN is an architecture which learns a one-to-one mapping between two domains. Each domain has an associated *discriminator*, while the mappings between these domains correspond to *generators*. The set of generators in CycleGAN is a collection of neural networks which is closed under composition, and whose inductive bias is increased by enforcing composition invariants, i.e. cycle-consistencies. The canonical example of this isomorphism used in [13] is that between the images of *horses* and *zebras*. Simply by changing the texture of the animal in such an image we can, approximately, map back and forth between these images.

Outline of the main contributions. In this paper we take a collection of abstractions known to deep learning practitioners and translate them into the language of category theory.

We package a notion of the interconnection of networks as a free category $\mathbf{Free}(G)$ on some graph G and specify any equivalences between networks as relations between morphisms as a quotient category $\mathbf{Free}(G)/\sim$. Given such a category – which we call a *schema*, inspired by [12] – we specify the architectures of its networks as a functor Arch. We reason about various other notions found in deep learning, such as datasets, embeddings, and parameter spaces. The training process is associated with an indexed family of functors $\{H_p : \mathbf{Free}(G) \to \mathbf{Set}\}_{i=1}^{T}$, where T is the number of training steps and p is some choice of a parameter for that architecture.

Analogous to standard neural networks – we start with a randomly initialized H_p and iteratively update it using gradient descent. Our optimization is guided by *two* objectives. These objectives arise as a natural generalization of those found in [13]. One of them is the adversarial objective – the minmax objective found in any Generative Adversarial Network. The other one is a generalization of the cycle-consistency loss which we call *path-equivalence loss*.

This approach yields useful insights and a large degree of generality: (i) it enables learning with unpaired data as it does not impose any constraints on ordering or pairing of the sets in a category, and (ii) although specialized to generative models in the domain of computer vision, this approach is domain-independent and general enough to hold in any domain of interest, such as sound, text, or video. This allows us to consider a subcategory of $\mathbf{Set}^{\mathbf{Free}(G)}$ as a space in which we can employ a gradient-based search. In other words, we *use the structure of categorical schemas as regularization during training*, such that the imposed relationships guide the learning process.

We show that for specific choices of $\mathbf{Free}(G)/\sim$ and the dataset we recover GAN [6] and CycleGAN [13]. Furthermore, a novel neural network architecture capable of learning to remove and insert objects into an image with unpaired data is proposed. We outline its categorical perspective and test it on the CelebA dataset.

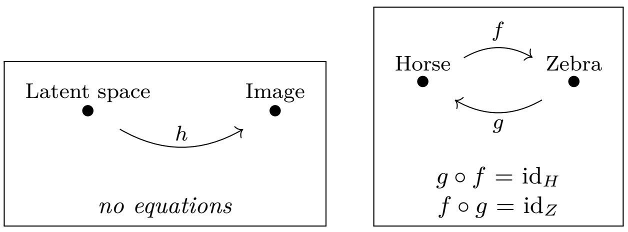
2 Categorical Deep Learning

Modern deep learning optimization algorithms can be framed as a gradient-based search in some function space Y^X, where X and Y are sets that have been endowed with extra structure. Given some sets of data points $D_X \subseteq X$, $D_Y \subseteq Y$, a typical approach for adding inductive bias relies on exploiting this extra structure associated to the data points embedded in those sets, or those sets themselves. This structure includes domain-specific features which can be exploited by various methods – convolutions for images, Fourier transform for audio, and specialized word embeddings for textual data.

In this section we develop the categorical tools to increase inductive bias of a model by enforcing the composition invariants of its constituent networks.

2.1 Model schema

Many deep learning models are complex systems, some comprised of several neural networks. Each neural network can be identified with domain X, codomain Y, and a *differentiable parametrized function* $X \to Y$. Given a *collection* of such networks, we use a directed multigraph to capture their interconnections. Each directed multigraph G gives rise to a corresponding free category on that graph $\mathbf{Free}(G)$. Based on this construction, Figure 2 shows the interconnection pattern for generators of two popular neural network architectures: GAN [6] and CycleGAN [13].



(a) GAN (b) CycleGAN

Figure 2: Bird's-eye view of two popular neural network models

Observe that CycleGAN has some additional properties imposed on it, specified by equations in Figure 2 (b). These are called cycle-consistency conditions and can roughly be stated as follows: given domains A and B considered as sets, $a \approx g(f(a)), \forall a \in A$ and $b \approx f(g(b)), \forall b \in B$. A particularly clear diagram of the cycle-consistency condition can be found in Figure 3 in [13].

Our approach involves a realization that cycle-consistency conditions *can be generalized* to path equivalence relations, or, in formal terms - a congruence relation. The condition $a \approx g(f(a)), \forall a \in A$ can be reformulated such that it does not refer to the elements of the set $a \in A$. By *eta-reducing* the equation we obtain $id_a = g \circ f$. Similar reformulation can be done for the other condition: $id_b = f \circ g$.

This allows us to package the newly formed equations as equivalence relations on the sets $\mathbf{Free}(G)(A, A)$ and $\mathbf{Free}(G)(B, B)$, respectively. This notion can be further packaged into a quotient category $\mathbf{Free}(G)/\sim$, together with the quotient functor $\mathbf{Free}(G) \xrightarrow{Q} \mathbf{Free}(G)/\sim$.

This formulation – as a free category on a graph G – represents the cornerstone of our approach. These schemas allow us to reason solely about the interconnections between various concepts, rather than jointly with functions, networks or other some other sets. All the other constructs in this paper are structure-preserving maps between categories whose domain, roughly, can be traced back to $\mathbf{Free}(G)$.

2.2 What is a neural network?

In computer science, the idea of a *neural network* colloquially means a number of different things. At a most fundamental level, it can be interpreted as a system of interconnected units called neurons, each of which has a firing threshold acting as an information filtering system. Drawing inspiration from biology, this perspective is thoroughly explored in literature. In many contexts we want to focus on the mathematical properties of a neural network and as such identify it with a function between sets $A \xrightarrow{f} B$. Those sets are often considered to have extra structure, such as those of Euclidean spaces or manifolds. Functions are then considered to be maps of a given differentiability class which preserve such structure. We also frequently reason about a neural network jointly with its parameter space P as a function of type $f : P \times A \to B$. For instance, consider a classifier in the context of supervised learning. A convolutional neural network whose input is a 32×32 `RGB` image and output is real number can be represented as a function with the following type: $\mathbb{R}^n \times \mathbb{R}^{32\times32\times3} \to \mathbb{R}$, for some $n \in \mathbb{N}$. In this case $\mathbb{R}^n$ represents the parameter space of this network.

The former ($A \to B$) and the latter ($P \times A \to B$) perspective on neural networks are related. Namely, consider some function space B^A. Any notion of smoothness in such a space is not well defined without any further assumptions on sets A or B. This is the reason deep learning employs a gradient-based search in such a space via a proxy function $P \times A \to B$. This function specifies an entire *parametrized family* of functions of type $A \to B$, because partial application of each $p \in P$ yields a function $f(p, -) : A \to B$. This choice of a parametrized family of functions is part of the *inductive bias* we are building into the training process. For example, in computer vision it is common to restrict the class of functions to those that can be modeled by convolutional neural networks.

With this in mind, we recall the model schema. For each morphism $A \to B$ in $\mathbf{Free}(G)$ we are interested in specifying a parametrized function $f : P \times A \to B$, i.e. a parametrized *family of functions* in **Set**. The function f describes a neural network architecture, and a choice of a partially applied $p \in P$ to f describes a choice of some parameter value for that specific architecture.

We capture the notion of parametrization with a category **Para** [4]. It is a strict symmetric monoidal category whose objects are Euclidean spaces and morphisms $\mathbb{R}^n \to \mathbb{R}^m$ are, roughly, differentiable functions of type $\mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^m$, for some p. We refer to $\mathbb{R}^p$ as the parameter space. Composition of morphisms in **Para** is defined in such a way that it explicitly keeps track of parameters. For more details, we refer the reader to [4].

A closely related construction we use is **Euc**, the strict symmetric monoidal category whose objects are finite-dimensional Euclidean spaces and morphisms are differentiable maps. A monoidal product on **Euc** is given by the cartesian product.

We package both of these notions – choosing an architecture and choosing parameters – into functors whose domain is $\mathbf{Free}(G)$ and codomains are **Para** and **Euc**, respectively.

2.3 Network architecture

We now formally specify *model architecture* as a functor. Observe that the action on morphisms is defined on the generators in $\mathbf{Free}(G)$.

Definition 1. *Architecture of a model is a functor* $\mathsf{Arch} : \mathbf{Free}(G) \to \mathbf{Para}$.

- *For each* $A \in Ob(\mathbf{Free}(G))$*, it specifies an Euclidean space* $\mathbb{R}^a$*;*
- *For each generating morphism* $A \xrightarrow{f} B$ *in* $\mathbf{Free}(G)$*, it specifies a morphism* $\mathbb{R}^a \xrightarrow{\mathsf{Arch}(f)} \mathbb{R}^b$ *which is a differentiable parametrized function of type* $\mathbb{R}^n \times \mathbb{R}^a \to \mathbb{R}^b$.

Given a non-trivial composite morphism $f = f_n \circ f_{n-1} \circ \cdots \circ f_1$ *in* $\mathcal{C}$*, the image of* f *under* Arch *is the composite of the image of each constituent:* $\mathsf{Arch}(f) = \mathsf{Arch}(f_n) \circ \mathsf{Arch}(f_{n-1}) \circ \cdots \circ \mathsf{Arch}(f_1)$. Arch *maps identities to the projection* $\pi_2 : I \times A \to A$.

The choice of architecture $\mathbf{Free}(G) \xrightarrow{\mathsf{Arch}} \mathbf{Para}$ goes hand in hand with the choice of an *embedding.*

Proposition 2. *An embedding is a functor* $|\mathbf{Free}(G)| \xrightarrow{E} \mathbf{Set}$ *which agrees with* Arch *on objects.*

Notice that the codomain of E is **Set**, rather than **Para**, as **Para** and **Euc** have the same objects and objects in **Euc** are just sets with extra structure.

Embedding E and Arch come up in two different scenarios. Sometimes we start out with a choice of architecture which then induces the embedding. In other cases, the embedding is given to us beforehand and it restricts the possible choice of architectures.

3 Parameter space

Each network architecture $f : \mathbb{R}^n \times \mathbb{R}^a \to \mathbb{R}^b$ comes equipped with its parameter space $\mathbb{R}^n$. Just as $\mathbf{Free}(G) \xrightarrow{\mathsf{Arch}} \mathbf{Para}$ is a categorical generalization of architecture, we now show there exists a categorical generalization of a parameter space. In this case – it is the parameter space of a functor. Before we move on to the main definition, we package the notion of parameter space of a function $f : \mathbb{R}^n \times \mathbb{R}^a \to \mathbb{R}^b$ into a simple function $\mathfrak{p}(f) = \mathbb{R}^n$.

Definition 3 (Functor parameter space)**.** *Let* $\mathsf{Gen}_{\mathbf{Free}(G)}$ *the set of generators in* $\mathbf{Free}(G)$*. The total parameter map* $\mathcal{P} : Ob(\mathbf{Para}^{\mathbf{Free}(G)}) \to Ob(\mathbf{Euc})$ *assigns to each functor* $\mathbf{Free}(G) \xrightarrow{\mathsf{Arch}} \mathbf{Para}$ *the product of the parameter spaces of all its generating morphisms:*

$$\mathcal{P}(\mathsf{Arch}) := \prod_{f \in \mathsf{Gen}_{\mathbf{Free}(G)}} \mathfrak{p}(\mathsf{Arch}(f))$$

Essentially, just as $\mathfrak{p}$ returns the parameter space of a function, $\mathcal{P}$ does the same for a *functor*.

We are now in a position to talk about parameter specification. Recall the non-categorical setting: given some network architecture $f : P \times A \to B$ and a choice of $p \in \mathfrak{p}(f)$ we can partially apply the parameter p to the network to get $f(p, -) : A \to B$. This admits a straightforward generalization to the categorical setting.

Definition 4 (Parameter specification)**.** *Parameter specification* PSpec *is a dependently typed function with the following signature:*

$$(\mathsf{Arch} : Ob(\mathbf{Para}^{\mathbf{Free}(G)})) \times \mathcal{P}(\mathsf{Arch}) \to Ob(\mathbf{Euc}^{\mathbf{Free}(G)}) \tag{1}$$

Given an architecture Arch *and a parameter choice* $(p_f)_{f \in \mathsf{Gen}_{\mathbf{Free}(G)}} \in \mathcal{P}(\mathsf{Arch})$ *for that architecture, it defines a choice of a functor in* $\mathbf{Euc}^{\mathbf{Free}(G)}$*. This functor acts on objects the same as* Arch*. On morphisms, it partially applies every* p_f *to the corresponding morphism* $\mathsf{Arch}(f) : \mathbb{R}^n \times \mathbb{R}^a \to \mathbb{R}^b$*, thus yielding* $f(p_f, -) : \mathbb{R}^a \to \mathbb{R}^b$ *in* $\mathbf{Euc}$*.*

Elements of $\mathbf{Euc}^{\mathbf{Free}(G)}$ will play a central role later on in the paper. These elements are functors which we will call *Models*. Given some architecture Arch and a parameter $p \in \mathcal{P}(\mathsf{Arch})$, a model $\mathbf{Free}(G) \xrightarrow{\mathsf{Model}_p} \mathbf{Euc}$ generalizes the standard notion of a model in machine learning – it can be used for inference and evaluated.

Analogous to database instances in [12], we call a *network instance* H_p the composition of some Model_p with the forgetful functor $\mathbf{Euc} \xrightarrow{U} \mathbf{Set}$. That is to say, a network instance is a functor $\mathbf{Free}(G) \xrightarrow{H_p} \mathbf{Set} := U \circ \mathsf{Model}_p$.

We shed some more light on these constructions using Figure 3.

$$\begin{tikzcd}
\mathbf{Free}(G) \arrow[r, "\mathsf{Arch}"] \arrow[dr, "\mathsf{Model}_p"] \arrow[ddr, "H_p"'] & \mathbf{Para} \\
 & \mathbf{Euc} \arrow[d, "U"] \\
 & \mathbf{Set}
\end{tikzcd}$$

Figure 3: $\mathbf{Free}(G)$ is the domain of three types of functors of interest: Arch, Model_p and H_p.

4 Data

We have described constructions which allow us to pick an architecture for a schema and consider its different models Model_p, each of them identified with a choice of a parameter $p \in \mathcal{P}(\mathsf{Arch})$. In order to understand how the optimization process is steered in updating the parameter choice for an architecture, we need to understand a vital component of any deep learning system – datasets themselves.

This necessitates that we also understand the relationship between datasets and the space they are embedded in. Recall the *embedding* functor and the notation $|\mathcal{C}|$ for the discretizaton of some category $\mathcal{C}$.

Definition 5. *Let* $|\mathbf{Free}(G)| \xrightarrow{E} \mathbf{Set}$ *be the embedding. A* ***dataset*** *is a subfunctor* $D_E : |\mathbf{Free}(G)| \to \mathbf{Set}$ *of* E*.* D_E *maps each object* $A \in Ob(\mathbf{Free}(G))$ *to a dataset* $D_E(A) := \{a_i\}_{i=1}^N \subseteq E(A)$*.*

Note that we refer to this functor in the singular, although it assigns a dataset to *each* object in $\mathbf{Free}(G)$. We also highlight that the domain of D_E is $|\mathbf{Free}(G)|$, rather than $\mathbf{Free}(G)$. We generally cannot provide an action on morphisms because datasets might be incomplete. Going back to the example with Horses and Zebras – a dataset functor on $\mathbf{Free}(G)$ in Figure 2 (b) maps Horse to the set of obtained horse images and Zebra to the set of obtained zebra images.

The subobject relation $D_E \subseteq E$ in Proposition 5 reflects an important property of data; we cannot obtain some data without it being in some shape or form, embedded in some larger space. Any obtained data thus implicitly fixes an embedding.

Observe that when we have a dataset in standard machine learning, we have a dataset *of something*. We can have a dataset of historical weather data, a dataset of housing prices in New York or a dataset of cat images. What ties all these concepts together is that each element a_i of some dataset $\{a_i\}_{i=1}^N$ is an instance of a more general concept. As a trivial example, every image in the dataset of horse images is a *horse*. The word *horse* refers to a more general concept and as such could be generalized from some of its instances which we *do not possess*. But all the horse images we possess are indeed an example of a horse. By considering everything to be embedded in some space $E(A)$ we capture this statement with the relation $\{a_i\}_{i=1}^N \subseteq \mathfrak{C}(A) \subseteq E(A)$. Here $\mathfrak{C}(A)$ is the set of all instances of some notion A which are embedded in $E(A)$. In the running example this corresponds to all images of horses in a given space, such as the space of all 64×64 `RGB` images. Obviously, the precise specification of $\mathfrak{C}(A)$ is unknown – as we cannot enumerate or specify the set of *all* horse images.

We use such calligraphy to denote this is an abstract concept. Despite the fact that its precise specification is unknown, we can still reason about its relationship to other structures. Furthermore, as it is the case with any abstract notion, there might be some edge cases or it might turn out that this concept is ambiguously defined or even inconsistent. Moreover, it might be possible to identify a dataset with multiple concepts; is a dataset of male human faces associated with the concept of male faces or is it a non-representative sample of all faces in general? We ignore these concerns and assume each dataset is a dataset of some well-defined, consistent and unambiguous concept. This does not change the validity of the rest of the formalism in any way as there exist plenty of datasets satisfying such a constraint.

Armed with intuition, we show this admits a generalization to the categorical setting. Just as $\{a_i\}_{i=1}^N \subseteq \mathfrak{C}(A) \subseteq E(A)$ are all subsets of $E(A)$ we might hypothesize the domain of $\mathfrak{C}$ is $|\mathbf{Free}(G)|$ and that $D_E \subseteq \mathfrak{C} \subseteq E$ are all subfunctors of E. However, just as we assign a set of all concept instances to *objects* in $\mathbf{Free}(G)$, we also assign a function between these sets to *morphisms* in $\mathbf{Free}(G)$. Unlike with datasets, this can be done because, by definition, these sets are not incomplete.

Definition 6. *Given a schema $\mathbf{Free}(G)/\sim$ and a dataset $|\mathbf{Free}(G)| \xrightarrow{D_E} \mathbf{Set}$, a **concept** associated with the dataset D_E embedded in E is a functor $\mathfrak{C} : \mathbf{Free}(G)/\sim \to \mathbf{Set}$ such that $D_E \subseteq \mathfrak{C} \circ I \subseteq E$. We say $\mathfrak{C}$ picks out sets of concept instances and functions between those sets.*

Another way to understand a concept $\mathbf{Free}(G)/\sim \xrightarrow{\mathfrak{C}} \mathbf{Set}$ is that it is required that a human observer can tell, for each $A \in Ob(\mathbf{Free}(G))$ and some $a \in E(A)$ whether $a \in \mathfrak{C}(A)$. Similarly for morphisms, a human observer should be able to tell if some function $\mathfrak{C}(A) \xrightarrow{f} \mathfrak{C}(B)$ is an image of some morphism in $\mathbf{Free}(G)/\sim$ under $\mathfrak{C}$.

For instance, consider the GAN schema in Figure 2 (a) where $\mathfrak{C}(\text{Image})$ is a set of all images of human faces embedded in some space such as $\mathbb{R}^{64\times64\times3}$. For each image in this space, a human observer should be able to tell if that image contains a face or not. We cannot enumerate such a set $\mathfrak{C}(\text{Image})$ or write it down explicitly, but we can easily tell if an image contains a given concept. Likewise, for a morphism in the CycleGAN schema (Figure 2 (b)), we cannot explicitly write down a function which transforms a horse into a zebra, but we can tell if some function did a good job or not by testing it on different inputs.

The most important thing related to this concept is that this represents the goal of our optimization process. Given a dataset $|\mathbf{Free}(G)| \xrightarrow{D_E} \mathbf{Set}$, want to extend it into a functor $\mathbf{Free}(G)/\sim \xrightarrow{\mathfrak{C}} \mathbf{Set}$, and actually *learn* its implementation.

4.1 Restriction of network instance to the dataset

We have seen how data is related to its embedding. We now describe the relationship between *network instances* and data.

Observe that network instance H_p maps each object $A \in Ob(\mathbf{Free}(G))$ to the entire embedding $H_{p_i}(A) = E(A)$, rather than just the concept $\mathfrak{C}(A)$. Even though we started out with an embedding $E(A)$, we want to narrow that embedding down just to the set of instances corresponding to some concept A.

For example, consider a diagram such as the one in Figure 2 (a). Suppose the result of a successful training was a functor $\mathbf{Free}(G) \xrightarrow{H} \mathbf{Set}$.

Suppose that the image of h is $H(h) : [0,1]^{100} \to [0,1]^{64\times64\times3}$. As such, our interest is mainly the restriction of $[0,1]^{64\times64\times3}$ to $\mathfrak{C}(\text{Image})$, the image of $[0,1]^{100}$ under $H(h)$, rather than the entire $[0,1]^{64\times64\times3}$. In the case of horses and zebras in Figure 2 (b), we are interested in a map $\mathfrak{C}(\text{Horse}) \to \mathfrak{C}(\text{Zebra})$ rather than a map $[0,1]^{64\times64\times3} \to [0,1]^{64\times64\times3}$. In what follows we show a construction which restricts some H_p to its smallest subfunctor which contains the dataset D_E. Recall the previously defined inclusion $|\mathbf{Free}(G)| \overset{I}{\hookrightarrow} \mathbf{Free}(G)$.

Definition 7. *Let $D_E : |\mathbf{Free}(G)| \to \mathbf{Set}$ be the* ***dataset****. Let $\mathbf{Free}(G) \xrightarrow{H_p} \mathbf{Set}$ be the network instance on $\mathbf{Free}(G)$. The* ***restriction*** *of H_p to D_E is a subfunctor of H_p defined as follows:*

$$I_{H_p} := \bigcap_{\{G \in Sub(H_p)) | D_E \subseteq G \circ I\}} G$$

where $Sub(H_p)$ is the set of subfunctors of H_p.

This definition is quite condensed so we supply some intuition. We first note that the meet is well defined because each G is a subfunctor of H. In Figure 4 we depict the newly defined constructions using a commutative diagram.

$$\begin{tikzcd}
& \mathbf{Free}(G) \arrow[dr, bend left=50, "H_p"] \arrow[dr, bend right=20, "I_{H_p}"'] \arrow[dr, phantom, "\subseteq"] & \\
|\mathbf{Free}(G)| \arrow[ur, hook, "I"] \arrow[rr, "D_E"] \arrow[rr, bend right=30, "E"'] \arrow[rr, phantom, bend right=15, "\subseteq"] & & \mathbf{Set}
\end{tikzcd}$$

Figure 4: The functor H_p contains I_{H_p} in such a way that D_E is a subfunctor of $I_{H_p} \circ I$.

It is useful to think of I_H as a restriction of H to the *smallest* functor which fits all data and mappings between the data. This means that I_{H_p} contains all data samples specified by D_E.

Corollary 8. *D_E is a subfunctor of $I_{H_p} \circ I$:*

Proof. This is straightforward to show, as I_{H_p} is the intersection of all subobjects of H which, when composed with the inclusion I contain D_E. Therefore $I_{H_p} \circ I$ contains D_E as well. □

5 Optimization

We now describe how data guides the search process. We identify the goal of this search with the concept functor $\mathbf{Free}(G)/\sim \xrightarrow{\mathfrak{C}} \mathbf{Set}$. This means that given a schema $\mathbf{Free}(G)/\sim$ and data $|\mathbf{Free}(G)| \xrightarrow{D_E} \mathbf{Set}$ we want to train some architecture and find a functor $\mathbf{Free}(G)/\sim \xrightarrow{H'} \mathbf{Set}$ that can be identified with $\mathfrak{C}$. Of course, unlike in the case of the concept $\mathfrak{C}$, the implementation of H' is something that will be known to us.

We now define the notion of a *task*.

Definition 9. *Let G be a directed multigraph and $\sim$ a congruence relation on $\mathbf{Free}(G)$. A task is a triple $(G, \sim, |\mathbf{Free}(G)| \xrightarrow{D_E} \mathbf{Set})$.*

In other words, a graph G and $\sim$ specify a schema $\mathbf{Free}(G)/\sim$ and a functor D_E specifies a dataset for that schema. Each dataset is a dataset *of something* and thus can be associated with a functor $\mathbf{Free}(G)/\sim \xrightarrow{\mathfrak{C}} \mathbf{Set}$. Moreover, recall that a dataset fixes an embedding E too, as $D_E \subseteq E$. This in turn also narrows our choice of architecture $\mathbf{Free}(G) \xrightarrow{\mathsf{Arch}} \mathbf{Para}$, as it has to agree with the embedding on objects. This situation fully reflects what happens in standard machine learning practice – a neural network $P \times A \to B$ has to be defined in such a way that its domain A and codomain B embed the datasets of all of its inputs and outputs, respectively.

Even though for the same schema $\mathbf{Free}(G)/\sim$ we might want to consider different datasets, we will always assume a chosen dataset corresponds to a single training goal $\mathfrak{C}$.

5.1 Optimization objectives

We generalize the training procedure described in [13] in a natural way, free of ad-hoc choices.

Suppose we have a task $(G, \sim, |\mathbf{Free}(G)| \xrightarrow{D_E} \mathbf{Set})$. After choosing an architecture $\mathbf{Free}(G) \xrightarrow{\mathsf{Arch}} \mathbf{Para}$ consistent with the embedding E and, hopefully, with the right inductive bias, we start with a randomly chosen parameter $\theta_0 \in \mathcal{P}(\mathsf{Arch})$. This amounts the choice of a specific $\mathbf{Free}(G) \xrightarrow{\mathsf{Model}_{\theta_0}} \mathbf{Euc}$. Using the loss function defined further down in this section, we partially differentiate each $f : \mathbb{R}^n \times \mathbb{R}^a \to \mathbb{R}^b \in \mathsf{Gen}_{\mathbf{Free}(G)}$ with respect to the corresponding p_f . We then obtain a new parameter value for that function using some update rule, such as Adam [9]. The product of these parameters for each of the generators $(p_f)_{f \in \mathsf{Gen}_{\mathbf{Free}(G)}}$ (Definition 3) defines a new parameter $\theta_1 \in \mathcal{P}(\mathsf{Arch})$ for the model $\mathsf{Model}_{\theta_1}$. This procedure allows us to iteratively update a given $\mathsf{Model}_{\theta_i}$ and as such fixes a linear order $\{\theta_0, \theta_1, \ldots, \theta_T\}$ on some subset of $\mathcal{P}(\mathsf{Arch})$.

The optimization objective for a model $\mathbf{Free}(G) \xrightarrow{\mathsf{Model}_\theta} \mathbf{Euc}$ and a task $(G, \sim, |\mathbf{Free}(G)| \xrightarrow{D_E} \mathbf{Set})$ is twofold. The total loss will be stated as a sum of the *adversarial loss* and a *path-equivalence* loss. We now describe both of these losses. As we slowly transition to standard machine learning lingo, we note that some of the notation here will be untyped due to the lack of the proper categorical understanding of these concepts.[1]

We start by assigning a discriminator to each object $A \in Ob(\mathbf{Free}(G))$ using the following function:

$$\mathbf{D} : (A : Ob(\mathbf{Free}(G))) \to \mathbf{Para}(\mathsf{Arch}(A), \mathbb{R})$$

This function assigns to each object $A \in Ob(\mathbf{Free}(G))$ a morphisms in **Para** such that its domain is that given by $\mathsf{Arch}(A)$. This will allow us to compose compatible generators and discriminators. For instance, consider $\mathsf{Arch}(A) = \mathbb{R}^a$. Discriminator $\mathbf{D}(A)$ is then a function of type $\mathbb{R}^q \times \mathbb{R}^a \to \mathbb{R} : \mathbf{Para}(\mathbb{R}^a, \mathbb{R})$, where $\mathbb{R}^q$ is the parameter space of the discriminator. As a slight abuse of notation – and to be more in line with machine learning notation – we will call $\mathbf{D}_A$ discriminator of the object A with some partially applied parameter value $\mathbf{D}(A)(p, -)$.

In the context of GANs, when we refer to a generator we refer to the image of a generating morphism in $\mathbf{Free}(G)$ under Arch. Similarly as with discriminators, a generator corresponding to a morphism $\mathbb{R}^a \xrightarrow{f} \mathbb{R}^b$ in **Para** with some partially applied parameter value will be denoted using $\mathbf{G}_f$.

The GAN minimax objective $\mathcal{L}^B_{GAN}$ for a generator $\mathbf{G}_f$ and a discriminator $\mathbf{D}_B$ is stated in Eq. (2). In this formulation we use the Wasserstein distance [3].

$$\begin{aligned} \mathcal{L}^B_{GAN}(\mathbf{G}_f, \mathbf{D}_B) := & \mathop{\mathbb{E}}_{b \sim D_E(B)} [\mathbf{D}_B(b)] \\ & - \mathop{\mathbb{E}}_{a \sim D_E(A)} [\mathbf{D}_B(\mathbf{G}_f(a))] \end{aligned} \tag{2}$$

The generator is trained to minimize the loss in the Eq. (2), while the discriminator is trained to maximize it.

The second component of the total loss is a generalization of *cycle-consistency loss* in CycleGAN [13], analogous to the generalization of the cycle-consistency condition in Section 2.1.

Definition 10. *Let $A \overset{f}{\underset{g}{\rightrightarrows}} B$ and suppose there exists a path equivalence $f = g$. For the equivalence $f = g$ and the model $\mathbf{Free}(G) \xrightarrow{\mathsf{Model}_i} \mathbf{Euc}$ we define a **path equivalence loss** $\mathcal{L}^{f,g}_{\sim}$ as follows:*

$$\mathcal{L}^{f,g}_{\sim} := \mathbb{E}_{a \sim D_E(A)} [||\mathsf{Model}_i(f)(a) - \mathsf{Model}_i(g)(a)||_1]$$

This enables us to state the total loss simply as a weighted sum of adversarial losses for all generators and path equivalence losses for all equations.

Definition 11. *The **total loss** is given as the sum of all adversarial and path equivalence losses:*

$$\mathcal{L}_i := \sum_{A \xrightarrow{f} B \in \mathsf{Gen}_{\mathbf{Free}(G)}} \mathcal{L}^B_{GAN}(\mathbf{G}_f, \mathbf{D}_B) + \gamma \sum_{f = g \in \sim} \mathcal{L}^{f,g}_{\sim}$$

where γ is a hyperparameter that balances between the adversarial loss and the path equivalence loss.

5.2 Path equivalence relations

There is one interesting case of the total loss – when the total path-equivalence loss is zero: $\sum_{f=g \in \sim} \mathcal{L}^{f,g}_{\sim} = 0$. This tells us that $H(f) = H(g)$ for all $f = g$ in $\sim$. In what follows we elaborate on what this means by recalling how $\mathbf{Free}(G)$ and $\mathbf{Free}(G)/\sim$ are related.

So far, we have been only considering schemas given by $\mathbf{Free}(G)$. This indeed is a limiting factor, as it assumes the categories of interest are only those without any imposed relations R between the generators G. One example of a schema *with* relations is the CycleGAN schema 2 (b) for

[1]Categorical formulation of the adversarial component of Generative Adversarial Networks is still an open problem. It seems to require nontrivial reformulations of existing constructions [4] and at least a partial integration of Open Games [5] into the framework of gradient-based optimization.

which fixing a functor $\mathbf{Free}(G)/\sim \to \mathbf{Set}$ requires that its image satisfies any relations imposed by $\mathbf{Free}(G)/\sim$. As neural network parameters usually are initialized randomly, any such image in **Set** will most surely not preserve such relations and thus will not be a proper functor whose domain is $\mathbf{Free}(G)/\sim$.

However, this construction is a functor if we consider its domain to be $\mathbf{Free}(G)$. Furthermore, assuming a successful training process whose end result is a path-equivalence relation preserving functor $\mathbf{Free}(G) \to \mathbf{Set}$, we show this induces an unique $\mathbf{Free}(G)/\sim \to \mathbf{Set}$ (Figure 5).

$$\begin{tikzcd}
\mathbf{Free}(G) \arrow[d, "Q"'] \arrow[dr, "H"] & \\
\mathbf{Free}(G)/\sim \arrow[r, dashed, "H'"] & \mathbf{Set}
\end{tikzcd}$$

Figure 5: Functor H which preserves path-equivalence relations factors uniquely through Q.

Theorem 12. *Let $Q : \mathbf{Free}(G) \to \mathbf{Free}(G)/\sim$ be the quotient functor and let $H : \mathbf{Free}(G) \to \mathbf{Set}$ be a path-equivalence relation preserving functor. Then there exists a unique functor $H' : \mathbf{Free}(G)/\sim \to \mathbf{Set}$ such that $H' \circ Q = H$.*

Proof. [11], Section 2.8., Proposition 1. □

Finding such a functor H is no easier task than finding a functor H'. However, this construction allow us to initially just guess a functor H_{θ_0}, since this initial choice will not have to preserve any relations. As training progresses and the path-equivalence loss of a network instance $\mathbf{Free}(G) \xrightarrow{H_\theta} \mathbf{Set}$ converges to zero, by Theorem 12 we show H_θ factors uniquely through $\mathbf{Free}(G) \xrightarrow{Q} \mathbf{Free}(G)/\sim$ via $\mathbf{Free}(G)/\sim \xrightarrow{H'} \mathbf{Set}$.

5.3 Functor space

Given an architecture Arch, each choice of $p \in \mathcal{P}(\mathsf{Arch})$ specifies a functor of type $\mathbf{Free}(G) \to \mathbf{Set}$. In this way exploration of the parameter space amounts to exploration of part of the functor category $\mathbf{Set}^{\mathbf{Free}(G)}$. Roughly stated, this means that a choice of an architecture adjoins a notion of *space* to the image of $\mathsf{PSpec}(\mathsf{Arch}, -)$ in the functor category $\mathbf{Set}^{\mathbf{Free}(G)}$. This space inherits all the properties of **Euc**.

By using gradient information to search the parameter space $\mathcal{P}(\mathsf{Arch})$, we are effectively using gradient information to search part of the functor space $\mathbf{Set}^{\mathbf{Free}(G)}$. Although we cannot explicitly explore just $\mathbf{Set}^{\mathbf{Free}(G)/\sim}$, we penalize the search method for veering into the parts of this space where the specified path equivalences do not hold. As such, the inductive bias of the model is increased without special constraints on the datasets or the embedding space - we merely require that the space is differentiable and that is has a sensible notion of distance.

Note that we do not claim inductive bias is *sufficient* to guarantee training convergence, merely that it is a useful regularization method applicable to a wide variety of situations. As categories can encode complex relationships between concepts and as functors map between categories in a structure-preserving way – this enables *structured learning* of concepts and their interconnections in a very general fashion.

6 Product task

We now present a choice of a dataset for the CycleGAN schema which makes up a novel task we will call *the product task*. The interpretation of this task comes in two flavors: as a simple change of dataset for the CycleGAN schema and as a method of composition and decomposition of images.

Just as we can take the product of two real numbers $a, b \in \mathbb{R}$ with a multiplication function $(a, b) \mapsto ab$, we show we can take a product of some two sets of images $A, B \in Ob(\mathbf{Set})$ with a neural network of type $A \times B \to C$. We will show $C \in Ob(\mathbf{Set})$ is a set of images which possesses all the properties of a categorical product.

In a cartesian category such as **Set** there already exists a notion of a categorical product – the cartesian product. Recall that the categorical product $A \times B$ is uniquely isomorphic to any other object AB which satisfies the universal property of the categorical product of objects A and B. This isomorphism will be central to the notion of the product task.

By *learning* the model $\mathbf{Free}(G) \xrightarrow{\mathsf{Model}} \mathbf{Euc}$ corresponding to the isomorphism $AB \cong A \times B$ we are also learning the projection maps θ_A and θ_B of AB. This follows from the universal property of the categorical product: $\pi_A \circ d = \theta_A$ and

$\pi_B \circ d = \theta_B$. Note how AB differs from a cartesian product. The domain of the corresponding projections θ_A and θ_B is not a simple pair of objects (a, b) and thus these projections cannot merely discard an element. θ_A needs to learn to remove A from a potentially complex domain. As such, this can be any complex, highly non-linear function which satisfies coherence conditions of a categorical product.

We will be concerned with supplying this new notion of the product AB with a dataset and learning the image of the isomorphism $AB \cong A \times B$ under $\mathbf{Free}(G) \xrightarrow{\mathsf{Model}} \mathbf{Set}$. We illustrate this on a concrete example. Consider a dataset A of images of human faces, a dataset B of images of glasses, and a dataset AB of people *wearing* glasses. Learning this isomorphism amounts to learning two things: (i) learning how to decompose an image of a person wearing glasses $(ab)_i$ into an image of a person a_j and image b_k of these glasses, and (ii) learning how to map this person a_j and some other glasses b_l into an image of a person a_j wearing glasses b_l. Generally, AB represents some sort of composition of objects A and B in the image space such that all information about A and B is preserved in AB. Of course, this might only be approximately true. Glasses usually cover a part of a face and sometimes its dark shades cover up the eyes – thus losing information about the eye color in the image and rendering the isomorphism invalid. However, in this paper we ignore such issues and assume that the networks $\mathsf{Arch}(d)$ can learn to unambiguously fill part of the face where the glasses were and that $\mathsf{Arch}(c)$ can learn to generate and superimpose the glasses on the relevant part of the face.

Even though we use the same CycleGAN schema from Figure 2 (b), we label one of its objects as AB and the other one as $A \times B$. Note that this does not change the schema itself, the labeling is merely for our convenience. The notion of a product or its projections is not captured in the schema itself. As schemas are merely categories presented with generators G and relations R, they lack the tools needed to encode a complex abstraction such as a universal construction. So how do we capture the notion of a product?

In this paper we frame this simply as a specific dataset functor, which we now describe. A dataset functor corresponding to the product task maps the object $A \times B$ in CycleGAN schema to a *cartesian product of two datasets*, $D_E(A \times B) = \{a_i\}_{i=0}^N \times \{b_j\}_{j=0}^M$. It maps the object AB to a dataset $\{(ab)_i\}_{i=0}^N$. In this case ab, a, and b are free to be any elements of datasets of a well-defined concept $\mathfrak{C}$. Although the difference between the product task and the CycleGAN task boils down to a different choice of a dataset functor, we note this is a key aspect which allows for a significantly different interpretation of the task semantics.

By considering A as some *image background* and B as the *object* which will be inserted, this allows us to interpret d and c as maps which *remove an object from the image* and *insert an object* in an image, respectively. This seems like a novel method of object generation and deletion with unpaired data, though we cannot claim to know the literature well enough to be sure.

7 Experiments

In this section we test whether the product task described in Section 6 can be trained in practice. In our experiments we use the CelebA dataset.

CelebFaces Attributes Dataset (CelebA) [10] is a large-scale face attributes dataset with more than 200000 celebrity images, each with 40 attribute annotations. Frequently used for image generation purposes, it fits perfectly into the proposed paradigm of the product task. The images in this dataset cover large pose variations and background clutter. The attribute annotations include "eyeglasses", "bangs", "pointy nose", "wavy hair" etc., as boolean flags for every image.

We used the attribute annotations to separate CelebA into two datasets. The dataset $D_E(AB)$ consists of images with the attribute "Eyeglasses", while the dataset $D_E(A)$ consists of all the other images.

Given that we could not obtain a dataset of images of *just glasses*, we set $D_E(B_Z) = [0,1]^{100}$ and add the subscript Z to B, as to make it more clear we are not generating images of this object. We refer to an element $z \in D_E(B_Z)$ as a *latent vector*, in line with machine learning terminology. This is a parametrization of all the missing information from A such that $A \times B_Z \cong AB$.

We investigate three things: (i) whether it is possible to *generate an image of a specific person wearing specific glasses*, (ii) whether we can *change* glasses that a person wears by changing the corresponding latent vector, and (iii) whether the same latent vector corresponds to the same glasses, irrespectively of the person we pair it with.

7.1 Results

In Figure 6 (left) we show the model learns the task (i): generating image of a specific person wearing glasses. Glasses are parametrized by the latent vector $z \in D_E(B_Z)$. The model learns to warp the glasses and put them in the right angle and size, based on the shape of the face. This can especially be seen in Figure 8, where some of the faces are seen from an angle, but glasses still blend in naturally. Figure 6 (right) shows the model learning task (ii): *changing* the glasses a person wears.

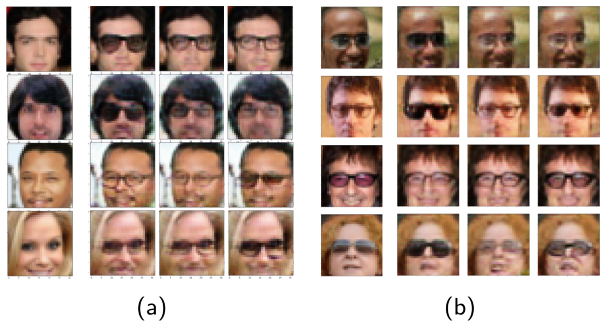


Figure 6: Parametrically *adding* glasses (a) and *changing* glasses (b) on a person's face. (a): the leftmost column shows a sample from the dataset $a_i \in D_E(A)$. Three rightmost columns show the result of $c(a_i, z_j)$, where $z_j \in D_E(B_Z)$ is a randomly sampled latent vector. (b): leftmost column shows a sample from the dataset $(ab)_i \in D_E(AB)$. Three rightmost columns show the image $c(\pi_A(d((ab_i))), z_j)$ which is the result of changing the glasses of a person. The latent vector $z_j \in D_E(B_Z)$ is randomly sampled.

Figure 7 shows the model can learn to *remove* glasses. Observe how in some cases the model did not learn to remove the glasses properly, as a slight outline of glasses can be seen.

An interesting test of the learned semantics can be done by checking if a specific randomly sampled latent vector z_j is consistent across different images. Does the resulting image of the application of $g(a_i, z_j)$, contain the same glasses as we vary the input image a_i? The results for the tasks (ii, iii) are shown in Figure 8. It shows how the network has learned to associate a specific vector z_j to a specific type of glasses and insert it in a natural way.

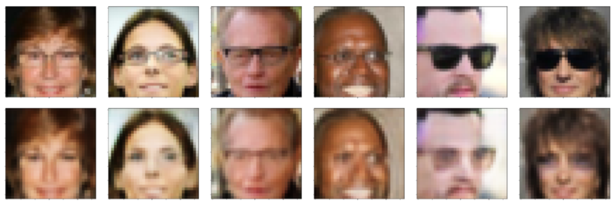


Figure 7: Top row shows samples $(ab)_i \in D_E(AB)$. Bottom row shows the result of a function $\pi_A \circ d : AB \to A$ which removes the glasses from the person.

We note low diversity in generated glasses and a slight loss in image quality, which is due to sub-optimal architecture choice for neural networks. Despite this, these experiments show that it is possible to train networks to (i) remove objects from, and (ii) parametrically insert objects into images in a *unsupervised, unpaired fashion*. Even though none of the networks were told that images contain people, glasses, or objects of any kind, we highlight that they learned to preserve all the main facial features.

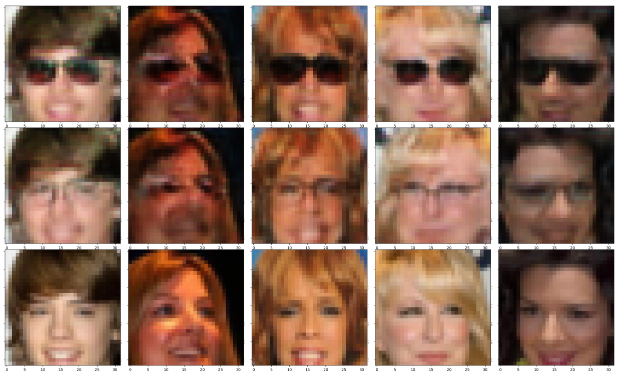


Figure 8: Bottom row shows true samples $a_i \in D_E(A)$. Top two rows show the image $c(a_i, z_j)$ of adding glases with *a specific latent vector* z_1 for the topmost row and z_2 for the middle row. Observe how the general style of the glasses stays the same in a given row, but gets adapted for every person that wears them.

8 From categorical databases to deep learning

The formulation presented in this paper bears a striking and unexpected similarity to Functorial Data Migration (FDM) [12]. Given a *categorical schema* $\mathbf{Free}(G)/\sim$ on some graph G, FDM defines a functor category $\mathbf{Set}^{\mathbf{Free}(G)/\sim}$ of database instance on that schema. The notion of *data in-*

tegrity is captured by *path equivalence relations* which ensure any specified "business rules" hold. The analogue of data integrity in neural networks is captured in the same way, first introduced in CycleGAN [13] as *cycle-consistency conditions*. The main difference between the approaches is that in this paper we do not start out with an implementation of the network instance functor, but rather we randomly initialize it and then *learn* it.

This shows that the underlying structures used for specifying data semantics for a given database systems are equivalent to the structures used to design data semantics which are possible to capture by training neural networks.

9 Conclusion and future work

In this paper we introduced a categorical formalism for training networks given by an arbitrary categorical schema. We showed a correspondence between categorical formulation of databases and neural network training and developed a rudimentary theory of *learning a specific class of functors using gradient descent*. Using the CelebA dataset we obtained experimental results and verified that semantic image manipulation can be carried out in a novel way.

We believe this to be one of the first steps exploring a rich connection between category theory and machine learning. It opens up interesting avenues of research and is seems to be deserving of further exploration.

References

[1] Amjad Almahairi, Sai Rajeswar, Alessandro Sordoni, Philip Bachman, and Aaron C. Courville. Augmented cyclegan: Learning many-to-many mappings from unpaired data. *CoRR*, abs/1802.10151, 2018. URL http://arxiv.org/abs/1802.10151.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *CoRR*, abs/1606.04474, 2016. URL http://arxiv.org/abs/1606.04474.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv e-prints*, art. arXiv:1701.07875, January 2017.

[4] Brendan Fong, David I. Spivak, and Rémy Tuyéras. Backprop as functor: A compositional perspective on supervised learning. *CoRR*, abs/1711.10455, 2017. URL http://arxiv.org/abs/1711.10455.

[5] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. *arXiv e-prints*, art. arXiv:1603.04641, March 2016.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[7] Jules Hedges. On compositionality. 2017. URL https://julesh.com/2017/04/22/on-compositionality/.

[8] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. *CoRR*, abs/1608.05343, 2016. URL http://arxiv.org/abs/1608.05343.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[10] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[11] Saunders MacLane. *Categories for the Working Mathematician*. Springer-Verlag, New York, 1971. Graduate Texts in Mathematics, Vol. 5.

[12] David I. Spivak. Functorial data migration. *CoRR*, abs/1009.1166, 2010. URL http://arxiv.org/abs/1009.1166.

[13] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017. URL http://arxiv.org/abs/1703.10593.