THIS IS YOUR MACHINE LEARNING SYSTEM?

YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.

WHAT IF THE ANSWERS ARE WRONG?

JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
Compositional Deep Learning

Bruno Gavranović

Faculty of Electrical Engineering and Computing (FER)
University of Zagreb, Croatia

bruno.gavranovic@fer.hr

December 18, 2018
Overview

- Usage of rudimentary category theory
Overview

- Usage of rudimentary category theory
- Neural networks
Usage of rudimentary category theory

Neural networks

- They’re compositional. You can *stack layers* and get better results
- They’re discovering (compositional) structures in data
Overview

- Usage of rudimentary category theory
- Neural networks
  - They’re compositional. You can *stack layers* and get better results
  - They’re discovering (compositional) structures in data
- Work in Progress
• Usage of rudimentary category theory
• Neural networks
  • They’re compositional. You can *stack layers* and get better results
  • They’re discovering (compositional) structures in data
• Work in Progress
• Experiments
We can generate completely realistic looking images
Space of all possible images

Bruno Gavranović  SYCO2  Compositional Deep Learning  December 18, 2018  6 / 36
Natural images form a low dimensional manifold in its embedding space
Generative Adversarial Networks

Training data
(e.g. 64x64x3≈12K dims)

Learning

Sampling

\( p(x) \)

\( \hat{p}(x) \)

But we have minimal control over the network output!

It’s possible to assign semantics to the network training procedure using the same schemas from Functorial Data Migration\(^1\)
It’s possible to assign semantics to the network training procedure using the same schemas from Functorial Data Migration¹

<table>
<thead>
<tr>
<th></th>
<th>Functorial Data Migration</th>
<th>Compositional Deep Learning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F : C \rightarrow -$</td>
<td>Set</td>
<td>Para</td>
</tr>
<tr>
<td>$F$ is</td>
<td>Fixed</td>
<td>Learned</td>
</tr>
</tbody>
</table>

¹ Bruno Gavranović  SYCO2  Compositional Deep Learning  December 18, 2018  8 / 36
Functorial data migration

- Categorical schema generated by a graph $G$ and a path equivalence relation: $\mathcal{C} := (G, \simeq)$

![Diagram showing a category with objects and morphisms]

Functorial data migration

- Categorical schema generated by a graph $G$ and a path equivalence relation: $\mathcal{C} := (G, \sim)$

![Diagram](image)

- A database instance is a functor $F : \mathcal{C} \rightarrow \text{Set}$

<table>
<thead>
<tr>
<th>Beatle</th>
<th>Played</th>
<th>Rock-and-roll instrument</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Lead guitar</td>
<td>Bass guitar</td>
</tr>
<tr>
<td>John</td>
<td>Rhythm guitar</td>
<td>Drums</td>
</tr>
<tr>
<td>Paul</td>
<td>Bass guitar</td>
<td>Keyboard</td>
</tr>
<tr>
<td>Ringo</td>
<td>Drums</td>
<td>Lead guitar</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rhythm guitar</td>
</tr>
</tbody>
</table>

Functorial data migration

- Categorical schema generated by a graph $G$ and a path equivalence relation: $C := (G, \simeq)$

$$\begin{array}{c}
\text{Beatle} \quad \text{Played} \quad \text{Rock-and-roll instrument}
\end{array}$$

- A database instance is a functor $F : C \rightarrow \text{Set}$

<table>
<thead>
<tr>
<th>Beatle</th>
<th>Played</th>
<th>Rock-and-roll instrument</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Lead guitar</td>
<td>Bass guitar</td>
</tr>
<tr>
<td>John</td>
<td>Rhythm guitar</td>
<td>Drums</td>
</tr>
<tr>
<td>Paul</td>
<td>Bass guitar</td>
<td>Keyboard</td>
</tr>
<tr>
<td>Ringo</td>
<td>Drums</td>
<td>Lead guitar</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rhythm guitar</td>
</tr>
</tbody>
</table>

- In databases, we have sets of data and clear mappings between them

1https://arxiv.org/abs/1803.05316
In machine learning all we have is plenty of data, but no known implementations of functions

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSample1</td>
<td>ExpectedOutput1</td>
</tr>
<tr>
<td>DataSample2</td>
<td>ExpectedOutput2</td>
</tr>
<tr>
<td>DataSample3</td>
<td>ExpectedOutput3</td>
</tr>
<tr>
<td>DataSample4</td>
<td>ExpectedOutput4</td>
</tr>
</tbody>
</table>
Paired

\[
\begin{align*}
\mathbf{x}_i & \quad \mathbf{y}_i \\
\{ & \quad , \\
\{ & \quad , \\
\{ & \quad , \\
\{ & \quad , \\
\end{align*}
\]

\[\text{https://arxiv.org/abs/1703.10593}\]
Paired

$\{x_i\}, \{y_i\}$

Unpaired

$X, Y$

1https://arxiv.org/abs/1703.10593
Style transfer

\[ X \quad Y \]
Style transfer

\[ X \xrightarrow{G} Y \]
Style transfer

\[ X \xrightarrow{G} Y \xleftarrow{F} \]
Style transfer
Style transfer

(a)

(b)

cycle-consistency loss

$G$

$F$

$X \rightarrow \hat{Y} \rightarrow \hat{X} \leftarrow Y$

$X \rightarrow \hat{Y} \rightarrow \hat{X} \leftarrow Y$

Bruno Gavranović SYCO2 Compositional Deep Learning December 18, 2018 12 / 36
Style transfer

(a)

(b)

cycle-consistency loss

(c)

cycle-consistency loss
CycleGAN

Monet $\leftrightarrow$ Photos
Monet $\rightarrow$ photo
photo $\rightarrow$ Monet

Zebras $\leftrightarrow$ Horses
zebra $\rightarrow$ horse
horse $\rightarrow$ zebra

Summer $\leftrightarrow$ Winter
summer $\rightarrow$ winter
winter $\rightarrow$ summer

Photograph $\rightarrow$
Monet
Van Gogh
Cezanne
Ukiyo-e
Backprop as Functor

- Compositional perspective on *supervised* learning
- Category of learners **Learn**
- Category of differentiable parametrized functions **Para**
Previous work

- Backprop as Functor
  - Compositional perspective on *supervised* learning
  - Category of learners **Learn**
  - Category of differentiable parametrized functions **Para**

- The Simple Essence of Automatic Differentiation
  - Compositional, *side-effect free* way of performing mode-independent automatic differentiation
Category of differentiable parametrized functions

Para:

Objects $a, b, c, ...$ are Euclidean spaces.

For each two objects $a, b$, we specify a set $\text{Para}(a, b)$ whose elements are differentiable functions of type $P \times A \rightarrow B$.

For every object $a$, we specify an identity morphism $\text{id}_a \in \text{Para}(a, a)$, a function of type $1 \times A \rightarrow A$, which is just a projection.

For every three objects $a, b, c$ and morphisms $f \in \text{Para}(A, B)$ and $g \in \text{Para}(B, C)$ one specifies a morphism $g \circ f \in \text{Para}(A, C)$ in the following way:

$\circ : (Q \times B \rightarrow C) \times (P \times A \rightarrow B) \rightarrow (P \times Q) \times A \rightarrow C$ (1)

$\circ (g, f) = \lambda ((p, q), a) \rightarrow g(q, f(p, a))$ (2)

Note: Coherence conditions are valid only up to isomorphism!

We can consider equivalence classes of morphisms or consider $\text{Para}$ as a bicategory.
Para:
- Objects $a, b, c, \ldots$ are Euclidean spaces
Para:

- Objects $a, b, c, ...$ are Euclidean spaces
- For each two objects $a, b$, we specify a set $\text{Para}(a, b)$ whose elements are differentiable functions of type $P \times A \to B$. 

Note: Coherence conditions are valid only up to isomorphism!
Category of differentiable parametrized functions

**Para:**

- Objects $a, b, c, ...$ are Euclidean spaces.
- For each two objects $a, b$, we specify a set $\text{Para}(a, b)$ whose elements are differentiable functions of type $P \times A \to B$.
- For every object $a$, we specify an identity morphism $id_a \in \text{Para}(a, a)$, a function of type $1 \times A \to A$, which is just a projection.
Para:

- Objects $a, b, c, \ldots$ are Euclidean spaces
- For each two objects $a, b$, we specify a set $\text{Para}(a, b)$ whose elements are differentiable functions of type $P \times A \rightarrow B$.
- For every object $a$, we specify an identity morphism $id_a \in \text{Para}(a, a)$, a function of type $1 \times A \rightarrow A$, which is just a projection.
- For every three objects $a, b, c$ and morphisms $f \in \text{Para}(A, B)$ and $g \in \text{Para}(B, C)$ one specifies a morphism $g \circ f \in \text{Para}(A, C)$ in the following way:
Para:

- Objects $a, b, c, \ldots$ are Euclidean spaces
- For each two objects $a, b$, we specify a set $\text{Para}(a, b)$ whose elements are differentiable functions of type $P \times A \to B$.
- For every object $a$, we specify an identity morphism $id_a \in \text{Para}(a, a)$, a function of type $1 \times A \to A$, which is just a projection.
- For every three objects $a, b, c$ and morphisms $f \in \text{Para}(A, B)$ and $g \in \text{Para}(B, C)$ one specifies a morphism $g \circ f \in \text{Para}(A, C)$ in the following way:

$$
\circ : (Q \times B \to C') \times (P \times A \to B) \to ((P \times Q) \times A \to C')
$$

$$
\circ(g, f) = \lambda((p, q), a) \to g(q, f(p, a))
$$
Category of differentiable parametrized functions

Para:

- Objects \(a, b, c, \ldots\) are Euclidean spaces
- For each two objects \(a, b\), we specify a set \(\text{Para}(a, b)\) whose elements are differentiable functions of type \(P \times A \to B\).
- For every object \(a\), we specify an identity morphism \(id_a \in \text{Para}(a, a)\), a function of type \(1 \times A \to A\), which is just a projection.
- For every three objects \(a, b, c\) and morphisms \(f \in \text{Para}(A, B)\) and \(g \in \text{Para}(B, C)\) one specifies a morphism \(g \circ f \in \text{Para}(A, C)\) in the following way:

\[
\circ : (Q \times B \to C') \times (P \times A \to B) \to ((P \times Q) \times A \to C')
\]

\[
\circ(g, f) = \lambda((p, q), a) \to g(q, f(p, a))
\]

- **Note:** Coherence conditions are valid only up to isomorphism!
Category of learners

Learn:
Let $A$ and $B$ be sets. A *supervised learning algorithm*, or simply *learner*, $A \rightarrow B$ is a tuple $(P, I, U, r)$ where $P$ is a set, and $I$, $U$, and $r$ are functions of types:

- $P : P$,
- $I : P \times A \rightarrow B$,
- $U : P \times A \times B \rightarrow P$,
- $r : P \times A \times B \rightarrow A$.

Update:

$$U_I(p, a, b) := p - \varepsilon \nabla p E_I(p, a, b)$$

Request

$$r_I(p, a, b) := f_a \left( \frac{1}{\alpha_B} \nabla_a E_I(p, a, b) \right),$$
Many overlapping notions

The update function \( U_I(p, a, b) := p - \varepsilon \nabla_p E_I(p, a, b) \) is computing two different things.

- It’s calculating the gradient \( p_g = \nabla_p E_I(p, a, b) \).
- It’s computing the parameter update by the rule of stochastic gradient descent: \((p, p_g) \mapsto p - \varepsilon p_g\).

Request function \( r \) in itself encodes the computation of \( \nabla_a E_I \).

Inside both \( r \) and \( U \) is embedded a notion of a cost function, which is fixed for all learners.
Many overlapping notions

- The update function \( U_I(p, a, b) := p - \varepsilon \nabla_p E_I(p, a, b) \) is computing \textit{two} different things.
  - It’s calculating the gradient \( p_g = \nabla_p E_I(p, a, b) \)
  - It’s computing the parameter update by the rule of stochastic gradient descent: \( (p, p_g) \mapsto p - \varepsilon p_g \).

- Request function \( r \) in itself encodes the computation of \( \nabla_a E_I \).

- Inside both \( r \) and \( U \) is embedded a notion of a cost function, which is fixed for all learners.

- \textbf{Problem}: These concepts are not separated into abstractions that reuse and compose well!
“Category of differentiable functions” is tricky to get right in a computational setting!
“Category of differentiable functions” is tricky to get right in a computational setting!

Implementing an efficient composable differentiation framework is more art than science.
“Category of differentiable functions” is tricky to get right in a computational setting!

Implementing an efficient composable differentiation framework is more art than science

Chain rule isn’t compositional \((g \circ f)'(x) = g'(f(x)) \cdot f'(x)\)
“Category of differentiable functions” is tricky to get right in a computational setting!

Implementing an efficient composable differentiation framework is more art than science

Chain rule isn’t compositional \((g \circ f)'(x) = g'(f(x)) \cdot f'(x)\)

- Derivative of the composition can’t be expressed only as a composition of derivatives!
“Category of differentiable functions” is tricky to get right in a computational setting!

Implementing an efficient composable differentiation framework is more art than science.

Chain rule isn’t compositional $$(g \circ f)'(x) = g'(f(x)) \cdot f'(x)$$

- Derivative of the composition can’t be expressed only as a composition of derivatives!

You need to store output of every function you evaluate.
“Category of differentiable functions” is tricky to get right in a computational setting!
Implementing an efficient composable differentiation framework is more art than science
Chain rule isn’t compositional \((g \circ f)'(x) = g'(f(x)) \cdot f'(x)\)
  - Derivative of the composition can’t be expressed only as a composition of derivatives!
You need to store output of every function you evaluate
Every deep learning framework has a carefully crafted implementation of side-effects
Automatic differentiation - category $\mathcal{D}$ of differentiable functions
• Automatic differentiation - category $\mathcal{D}$ of differentiable functions
• Morphism $A \to B$ is a function of type $a \to b \times (a \to b)$
Automatic differentiation - category $D$ of differentiable functions

Morphism $A \to B$ is a function of type $a \to b \times (a \to b)$

Composition: $g \circ f = \lambda a \to \text{let}(b, f'(a), (c, g') = g(b) \quad \text{in}(c, g' \circ f'))$
Automatic differentiation - category $\mathcal{D}$ of differentiable functions

**Morphism** $A \rightarrow B$ is a function of type $a \rightarrow b \times (a \rightarrow b)$

**Composition:** $g \circ f = \lambda a \rightarrow \text{let}(b, f') = f(a), (c, g') = g(b) \text{ in } (c, g' \circ f')$

**Structure for splitting and joining wires**
Automatic differentiation - category $\mathbb{D}$ of differentiable functions

Morphism $A \rightarrow B$ is a function of type $a \rightarrow b \times (a \rightarrow b)$

Composition: $g \circ f = \lambda a \rightarrow \text{let} (b, f') = f(a), (c, g') = g(b) \quad \text{in} (c, g' \circ f')$

Structure for splitting and joining wires

Generalization to more than just linear maps
Automatic differentiation - category $\mathcal{D}$ of differentiable functions

Morphism $A \to B$ is a function of type $a \to b \times (a \to b)$

Composition: $g \circ f = \lambda a \to \text{let}(b, f') = f(a), (c, g') = g(b) \quad \text{in}(c, g' \circ f')$

Structure for splitting and joining wires

Generalization to more than just linear maps

- Forward-mode automatic differentiation
- Reverse-mode automatic differentiation
- Backpropagation - $\mathcal{D}_{\text{Dual} \to +}$
BackpropFunctor + SimpleAD

- BackpropFunctor doesn’t mention categorical differentiation
BackpropFunctor + SimpleAD

- BackpropFunctor doesn’t mention categorical differentiation
- SimpleAD doesn’t talk about learning itself
BackpropFunctor + SimpleAD

- BackpropFunctor doesn’t mention categorical differentiation
- SimpleAD doesn’t talk about learning itself
- Both are talking about similar concepts
BackpropFunctor doesn’t mention categorical differentiation
SimpleAD doesn’t talk about learning itself
Both are talking about similar concepts
For each $P \times A \to B$ in $\text{Hom}(a, b)$ in $\text{Para}$, we’d like to specify a set of functions of type $P \times A \to B \times ((P \times A) \to B)$ instead of just $P \times A \to B$
BackpropFunctor + SimpleAD

- BackpropFunctor doesn’t mention categorical differentiation
- SimpleAD doesn’t talk about learning itself
- Both are talking about similar concepts
- For each \( P \times A \rightarrow B \) in \( Hom(a, b) \) in Para, we’d like to specify a set of functions of type \( P \times A \rightarrow B \times ((P \times A) \rightarrow B) \) instead of just \( P \times A \rightarrow B \)
- Separate the structure needed for parametricity and structure needed for composable differentiability

Bruno Gavranović SYCO2 Compositional Deep Learning December 18, 2018 20 / 36
BackpropFunctor + SimpleAD

- BackpropFunctor doesn’t mention categorical differentiation
- SimpleAD doesn’t talk about learning itself
- Both are talking about similar concepts
- For each $P \times A \to B$ in $\text{Hom}(a, b)$ in Para, we’d like to specify a set of functions of type $P \times A \to B \times ((P \times A) \to B)$ instead of just $P \times A \to B$
- Separate the structure needed for parametricity and structure needed for composable differentiability
- Solution: ?
Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$.
Main result

- Specify the semantics of your datasets with a categorical schema $C := (G, \simeq)$
Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$.

- Learn a functor $P: \mathcal{C} \rightarrow \text{Para}$
- Start with a functor $\text{Free}(G) \rightarrow \text{Para}$
- Iteratively update it using samples from your datasets
- The learned functor will also preserve $\simeq$

Novel regularization mechanism for neural networks.
Main result

- Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$
- Learn a functor $P : \mathcal{C} \to \text{Para}$

\[
\begin{align*}
Horse & \Rightarrow Zebra \\
\downarrow^f & \quad \downarrow^g \\
\text{Horse} & \quad \text{Zebra}
\end{align*}
\]

\[
f \cdot g = \text{id}_h, \quad g \cdot f = \text{id}_z
\]
Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$

Learn a functor $P : \mathcal{C} \to \text{Para}$

- Start with a functor $\text{Free}(G) \to \text{Para}$
Main result

- Specify the semantics of your datasets with a categorical schema \( C := (G, \simeq) \)
- Learn a functor \( P : C \to \text{Para} \)
  - Start with a functor \( \text{Free}(G) \to \text{Para} \)
  - Iteratively update it using samples from your datasets
  - The learned functor will also preserve \( \simeq \)

Novel regularization mechanism for neural networks.
Main result

- Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$
- Learn a functor $P : \mathcal{C} \to \text{Para}$
  - Start with a functor $\text{Free}(G) \to \text{Para}$
  - Iteratively update it using samples from your datasets
  - The learned functor will also preserve $\simeq$
- Novel regularization mechanism for neural networks.

\[ \begin{align*}
\text{Horse} & \xrightarrow{f} \text{Zebra} \\
\text{Zebra} & \xleftarrow{g} \text{Horse}
\end{align*} \]

\[ f \circ g = \text{id}_h, \quad g \circ f = \text{id}_z \]

\[ \mathbb{R}^{64 \times 64 \times 3} \xrightarrow{P\, f} \mathbb{R}^{64 \times 64 \times 3} \]

\[ \mathbb{R}^{64 \times 64 \times 3} \xleftarrow{P\, g} \mathbb{R}^{64 \times 64 \times 3} \]
Start with a functor $\text{Free}(G) \rightarrow \text{Para}$
Start with a functor $\text{Free}(G) \to \text{Para}$

- Specify how it acts on objects
Start with a functor $\text{Free}(G) \rightarrow \text{Para}$

- Specify how it acts on objects
- Start with randomly initialized morphisms
Start with a functor \( \text{Free}(G) \rightarrow \text{Para} \)

- Specify how it acts on objects
- Start with randomly initialized morphisms
  - Every morphism in \( \text{Para} \) is a function parametrized by some \( P \)
Start with a functor \( \text{Free}(G) \rightarrow \text{Para} \)

- Specify how it acts on objects
- Start with randomly initialized morphisms
  - Every morphism in \( \text{Para} \) is a function parametrized by some \( P \)
  - Initializing \( P \) randomly => “initializing” a morphism
Start with a functor \( \text{Free}(G) \rightarrow \text{Para} \)

- Specify how it acts on objects
- Start with randomly initialized morphisms
  - Every morphism in \( \text{Para} \) is a function parametrized by some \( P \)
  - Initializing \( P \) randomly => “initializing” a morphism

- Get data samples \( d_a, d_b, \ldots \) corresponding to every object in \( C \) and in every iteration:
Start with a functor \( \text{Free}(G) \to \text{Para} \)
- Specify how it acts on objects
- Start with randomly initialized morphisms
  - Every morphism in \( \text{Para} \) is a function parametrized by some \( P \)
  - Initializing \( P \) randomly => “initializing” a morphism

Get data samples \( d_a, d_b, ... \) corresponding to every object in \( C \) and in every iteration:
- For every morphism \( (f : A \to B) \) in the transitive reduction of morphisms in \( C \), find \( P f \) and minimize the distance between \( (P f)(d_a) \) and the corresponding image manifold
Start with a functor \( \text{Free}(G) \rightarrow \text{Para} \)

- Specify how it acts on objects
- Start with randomly initialized morphisms
  - Every morphism in \( \text{Para} \) is a function parametrized by some \( P \)
  - Initializing \( P \) randomly => "initializing" a morphism

Get data samples \( d_a, d_b, \ldots \) corresponding to every object in \( C \) and in every iteration:

- For every morphism \( (f : A \rightarrow B) \) in the transitive reduction of morphisms in \( C \), find \( Pf \) and minimize the distance between \( (Pf)(d_a) \) and the corresponding image manifold
- For all path equations from \( A \rightarrow B \) where \( f = g \), compute both \( f(R_a) \) and \( g(R_a) \). Calculate the distance \( d = ||f(R_a) - g(R_a)|| \). Minimize \( d \) and update all parameters of \( f \) and \( g \).
Start with a functor \textbf{Free}(G) \rightarrow \textbf{Para}

- Specify how it acts on objects
- Start with randomly initialized morphisms
  - Every morphism in \textbf{Para} is a function parametrized by some $P$
  - Initializing $P$ randomly \Rightarrow “initializing” a morphism

Get data samples $d_a, d_b, \ldots$ corresponding to every object in $C$ and in every iteration:

- For every morphism $(f : A \rightarrow B)$ in the transitive reduction of morphisms in $C$, find $Pf$ and minimize the distance between $(Pf)(d_a)$ and the corresponding image manifold
- For all \textbf{path equations} from $A \rightarrow B$ where $f = g$, compute both $f(R_a)$ and $g(R_a)$. Calculate the distance $d = ||f(R_a) - g(R_a)||$. Minimize $d$ and update all parameters of $f$ and $g$.

The path equation regularization term forces the optimization procedure to select functors which preserve the path equivalence relation and, thus, $C$
Some possible schemas
Some possible schemas

- This procedure generalizes several existing network architectures
Some possible schemas

- This procedure generalizes several existing network architectures
- But it also allows us to ask, what other interesting schemas are possible?
Some possible schemas

Figure: GAN

Figure: CycleGAN

Figure: Equalizer

Figure: Product
Given two networks $h, g : B \to C$, find a subset $B' \subseteq B$ such that
$$B' = \{b \in B \mid h(b) = g(b)\}$$
Consider two sets of images

- Left: Background of color X with a circle with fixed size and position of color Y
- Right: Background of color Z
Product schema

Same learning algorithm can learn to remove both types of objects

Bruno Gavranović SYCO2
Product schema

\[ f \cdot g = \text{id}_A \]
\[ g \cdot f = \text{id}_{B \times C} \]
Product schema

\[ f \cdot g = \text{id}_A \]
\[ g \cdot f = \text{id}_{B \times C} \]
Product schema

\[ f \cdot g = \text{id}_A \]
\[ g \cdot f = \text{id}_{B \times C} \]

- Same learning algorithm can learn to remove both types of objects
Experiments

- CelebA dataset of 200K images of human faces

- Eyeglasses
- Bangs
- Pointy Nose
- Oval Face
- Wearing Hat
- Wavy Hair
- Mustache
- Smiling
Experiments

- CelebA dataset of 200K images of human faces

- Conveniently, there is a “glasses” annotation
Experiments

\[ \text{PC} := \mathbb{R}^{32 \times 32 \times 3} \xrightarrow{Pf} \mathbb{R}^{32 \times 32 \times 3 \times 3} \xrightarrow{Pg} \mathbb{R}^{32 \times 32 \times 3 \times 100} \]

\[ f \cdot g = \text{id}_H \]
\[ g \cdot f = \text{id}_Z \]

- Collection of neural networks with total 40m parameters
- 7h training on a GeForce GTX 1080
- Successful results
Figure: Same image, different Z vector
Figure: Same Z vector, different image
Figure: Top row: original image, bottom row: Removed glasses
Specify a collection of neural networks which are closed under composition
Conclusions

- Specify a collection of neural networks which are closed under composition
- Specify composition invariants
Conclusions

- Specify a collection of neural networks which are closed under composition
- Specify composition invariants
- Given the right data and parametrized functions of sufficient complexity, it’s possible to train them with the right inductive bias
Conclusions

- Specify a collection of neural networks which are closed under composition
- Specify composition invariants
- Given the right data and parametrized functions of sufficient complexity, it’s possible to train them with the right inductive bias
- Common language to talk about semantics of data and training procedure
Future work

- This is still rough around the edges
Future work

- This is still rough around the edges
- What other schemas can we think of?
Future work

- This is still rough around the edges
- What other schemas can we think of?
- Can we quantify type of information we’re giving to the network using these schemas?
Future work

- This is still rough around the edges
- What other schemas can we think of?
- Can we quantify type of information we’re giving to the network using these schemas?
- Do data migration functors make sense in the context of neural networks?
Future work

- This is still rough around the edges
- What other schemas can we think of?
- Can we quantify type of information we’re giving to the network using these schemas?
- Do data migration functors make sense in the context of neural networks?
- Can game-theoretic properties of Generative Adversarial Networks be expressed categorically?
Future work

- This is still rough around the edges
- What other schemas can we think of?
- Can we quantify type of information we’re giving to the network using these schemas?
- Do data migration functors make sense in the context of neural networks?
- Can game-theoretic properties of Generative Adversarial Networks be expressed categorically?
- Coding these ideas in Idris
Thank you!

Bruno Gavranović
Faculty of Electrical Engineering and Computing
University of Zagreb
bruno.gavranovic@fer.hr

Feel free to drop me an email with any questions!