
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.
DATA
ANSWERS

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

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

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

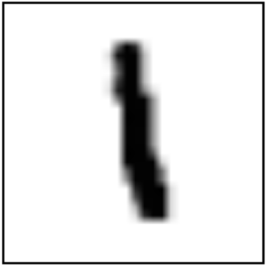
Overview

- 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
- Experiments

Generative modelling - State of the art - 2018

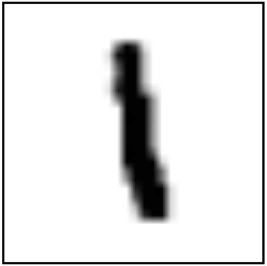
We can generate completely realistic looking images

Space of all possible images



$$\cong \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

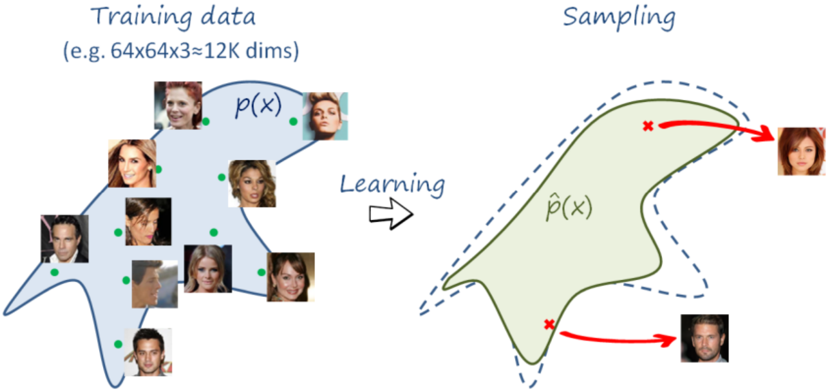
Space of all possible images



$$\cong \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\ 0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\ 0&0&0&0&0&0&0&1&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\ 0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$$

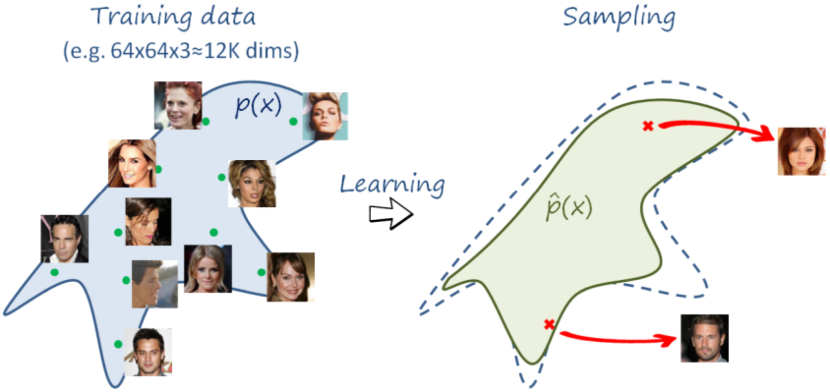
- Natural images form a low dimensional manifold in its embedding space

Generative Adversarial Networks



[0] `http://dl-ai.blogspot.com/2017/08/gan-problems.html`

Generative Adversarial Networks



But we have minimal control over the network output!

[0] http://dl-ai.blogspot.com/2017/08/gan-problems.html

Claim

It's possible to assign semantics to the network training procedure using the same schemas from Functorial Data Migration[1]

1

Claim

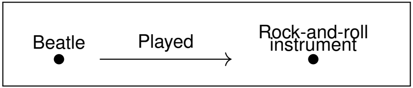
It's possible to assign semantics to the network training procedure using the same schemas from Functorial Data Migration[1]

	Functorial Data Migration	Compositional Deep Learning
$F : \mathcal{C} \to -$	$\mathbf{Set}$	$\mathbf{Para}$
F is	Fixed	Learned

1

Functorial data migration

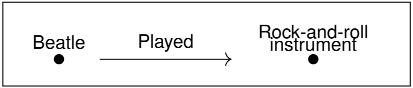
- Categorical schema generated by a graph G and a path equivalence relation: $\mathcal{C} := (G, \simeq)$



[1] https://arxiv.org/abs/1803.05316

Functorial data migration

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



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

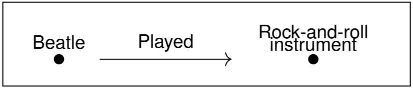
Beatle	**Played**
George	Lead guitar
John	Rhythm guitar
Paul	Bass guitar
Ringo	Drums

Rock-and-roll instrument
Bass guitar
Drums
Keyboard
Lead guitar
Rhythm guitar

[1] `https://arxiv.org/abs/1803.05316`

Functorial data migration

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



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

Beatle	**Played**
George	Lead guitar
John	Rhythm guitar
Paul	Bass guitar
Ringo	Drums

Rock-and-roll instrument
Bass guitar
Drums
Keyboard
Lead guitar
Rhythm guitar

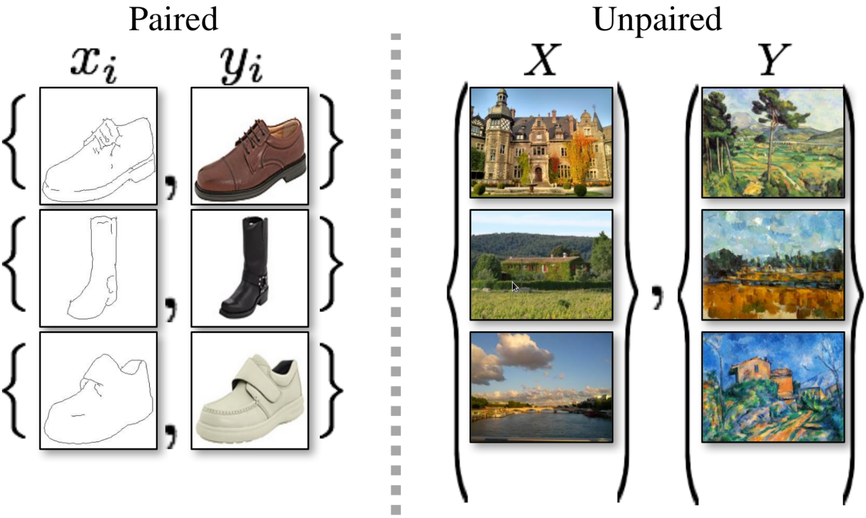
- In databases, we have sets of data and clear mappings between them

[1] https://arxiv.org/abs/1803.05316

- In machine learning all we have is plenty of data, but no known implementations of functions

Input	**Output**
DataSample1	ExpectedOutput1
DataSample2	ExpectedOutput2
DataSample3	ExpectedOutput3
DataSample4	ExpectedOutput4

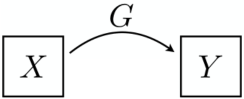
[1] https://arxiv.org/abs/1703.10593



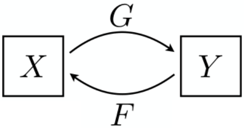
[1] https://arxiv.org/abs/1703.10593

Style transfer

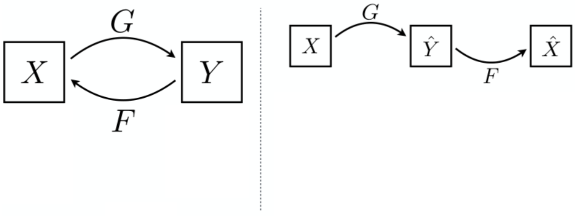
Style transfer



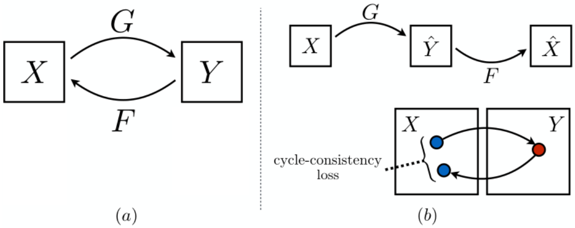
Style transfer



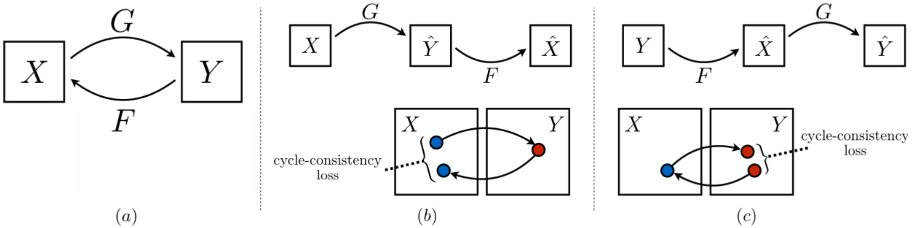
Style transfer



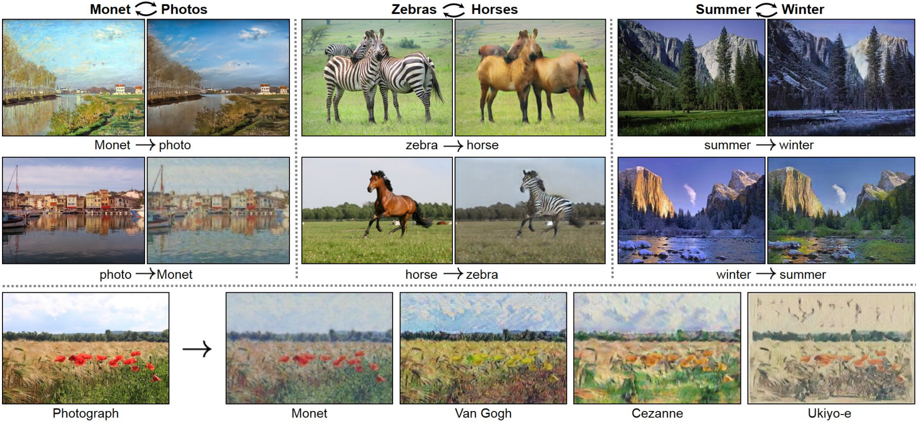
Style transfer



Style transfer



CycleGAN


Monet ⇄ Photos
Zebras ⇄ Horses
Summer ⇄ Winter
Monet → photo
zebra → horse
summer → winter
photo → Monet
horse → zebra
winter → summer
Photograph
→
Monet
Van Gogh
Cezanne
Ukiyo-e

Previous work

- Backprop as Functor
  - Compositional perspective on *supervised* learning
  - Category of learners $\mathbf{Learn}$
  - Category of differentiable parametrized functions $\mathbf{Para}$

Previous work

- Backprop as Functor
  - Compositional perspective on *supervised* learning
  - Category of learners $\mathbf{Learn}$
  - Category of differentiable parametrized functions $\mathbf{Para}$
- The Simple Essence of Automatic Differentiation
  - Compositional, *side-effect free* way of performing mode-independent automatic differentiation

Category of differentiable parametrized functions

$\mathbf{Para}$:

Category of differentiable parametrized functions

Para:

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

Category of differentiable parametrized functions

$\mathbf{Para}$:

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

Category of differentiable parametrized functions

$\mathbf{Para}$:

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

Category of differentiable parametrized functions

$\mathbf{Para}$:

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

Category of differentiable parametrized functions

$\mathbf{Para}$:

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

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

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

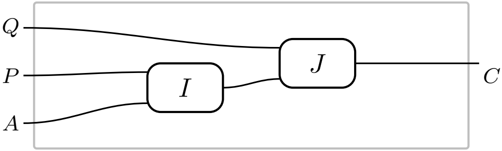
Category of differentiable parametrized functions

$\mathbf{Para}$:

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

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

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



- **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*, $A \to B$ is a tuple (P, I, U, r) where P is a set, and I, U, and r are functions of types:

$$\begin{aligned} P &\colon P, \\ I &\colon P \times A \to B, \\ U &\colon P \times A \times B \to P, \\ r &\colon P \times A \times B \to A. \end{aligned}$$

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 $U_I(p, a, b) \coloneqq p - \varepsilon \nabla_p E_I(p, a, b)$ is computing *two* different things.
  - It's calcuating 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 $U_I(p, a, b) \coloneqq p - \varepsilon \nabla_p E_I(p, a, b)$ is computing *two* different things.
  - It's calcuating 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.
- **Problem:** These concepts are not separated into abstractions that reuse and compose well!

The Simple Essence of Automatic Differentiation

- “Category of differentiable functions” is tricky to get right in a computational setting!

The Simple Essence of Automatic Differentiation

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

The Simple Essence of Automatic Differentiation

- "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 $(g \circ f)'(x) = g'(f(x)) \cdot f'(x)$

The Simple Essence of Automatic Differentiation

- “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!

The Simple Essence of Automatic Differentiation

- "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

The Simple Essence of Automatic Differentiation

- “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

The Simple Essence of Automatic Differentiation

- Automatic differentiation - category $\mathbf{D}$ of differentiable functions

The Simple Essence of Automatic Differentiation

- Automatic differentiation - category $\mathbf{D}$ of differentiable functions
- Morphism $A \to B$ is a function of type $a \to b \times (a \multimap b)$

The Simple Essence of Automatic Differentiation

- Automatic differentiation - category $\mathbf{D}$ of differentiable functions
- Morphism $A \to B$ is a function of type $a \to b \times (a \multimap b)$
- Composition: $g \circ f = \lambda a \to \mathrm{let}(b, f') = f(a), (c, g') = g(b) \quad \mathrm{in}(c, g' \circ f')$

The Simple Essence of Automatic Differentiation

- Automatic differentiation - category $\mathbf{D}$ of differentiable functions
- Morphism $A \to B$ is a function of type $a \to b \times (a \multimap b)$
- 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

The Simple Essence of Automatic Differentiation

- Automatic differentiation - category $\mathbf{D}$ of differentiable functions
- Morphism $A \to B$ is a function of type $a \to b \times (a \multimap b)$
- Composition: $g \circ f = \lambda a \to \mathrm{let}(b, f') = f(a), (c, g') = g(b) \quad \mathrm{in}(c, g' \circ f')$
- Structure for splitting and joining wires
- Generalization to more than just linear maps

The Simple Essence of Automatic Differentiation

- Automatic differentiation - category $\mathbf{D}$ of differentiable functions
- Morphism $A \to B$ is a function of type $a \to b \times (a \multimap b)$
- Composition: $g \circ f = \lambda a \to \text{let}(b, f') = f(a), (c, g') = g(b) \quad \textbf{in}(c, g' \circ f')$
- Structure for splitting and joining wires
- Generalization to more than just linear maps
  - Forward-mode automatic differentiation
  - Reverse-mode automatic differentiation
  - Backpropagation - $\mathbf{D}_{\mathbf{Dual}_{\to +}}$

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 + 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 $Hom(a, b)$ in $\mathbf{Para}$, we'd like to specify a set of functions of type $P \times A \to B \times ((P \times A) \multimap B)$ 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 $P \times A \to B$ in $Hom(a, b)$ in $\mathbf{Para}$, we'd like to specify a set of functions of type $P \times A \to B \times ((P \times A) \multimap B)$ instead of just $P \times A \to B$
- Separate the structure needed for parametricity and structure needed for composable differentiability

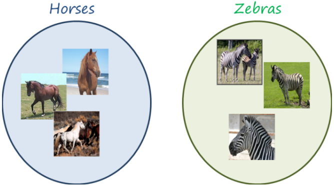
BackpropFunctor + SimpleAD

- 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 $Hom(a, b)$ in $\mathbf{Para}$, we'd like to specify a set of functions of type $P \times A \to B \times ((P \times A) \multimap B)$ instead of just $P \times A \to B$
- Separate the structure needed for parametricity and structure needed for composable differentiability
- Solution: ?

Main result

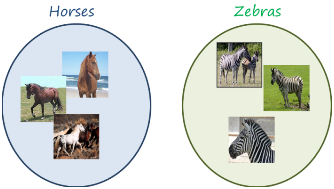
- Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$

Main result

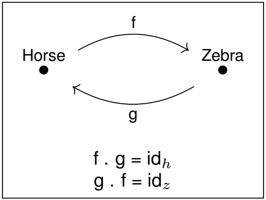


- Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$

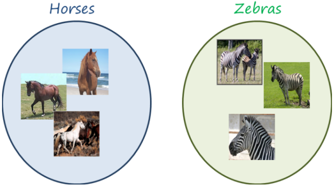
Main result



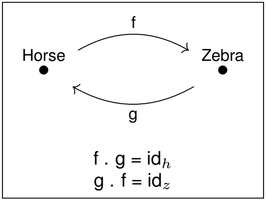
- Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$



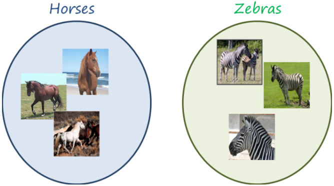
Main result

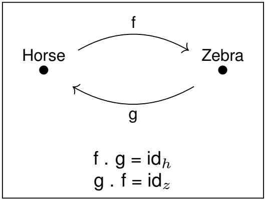


- Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$
- Learn a functor $P : \mathcal{C} \to \mathbf{Para}$



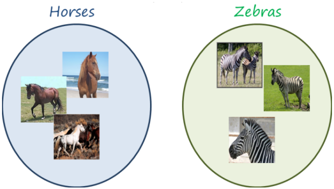
Main result



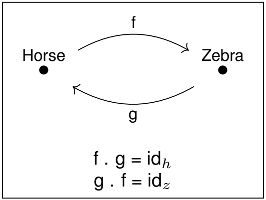


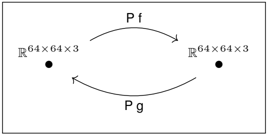
- Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$
- Learn a functor $P : \mathcal{C} \to \mathbf{Para}$
  - Start with a functor $\mathbf{Free}(\mathbf{G}) \to \mathbf{Para}$

Main result

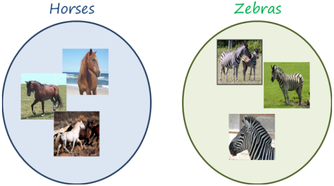


- Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$
- Learn a functor $P : \mathcal{C} \to \mathbf{Para}$
    - Start with a functor $\mathbf{Free(G)} \to \mathbf{Para}$
    - Iteratively update it using samples from your datasets
    - The learned functor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

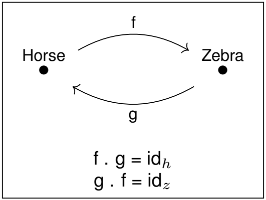


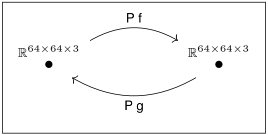


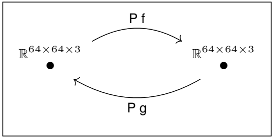
Main result



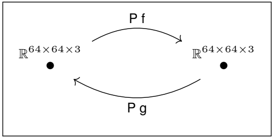
- Specify the semantics of your datasets with a categorical schema $\mathcal{C} := (G, \simeq)$
- Learn a functor $P : \mathcal{C} \to \mathbf{Para}$
  - Start with a functor $\mathbf{Free(G)} \to \mathbf{Para}$
  - Iteratively update it using samples from your datasets
  - The learned functor will also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



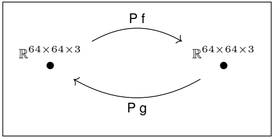




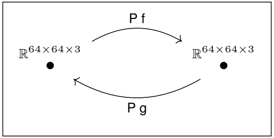
- Start with a functor $\mathbf{Free(G)} \to \mathbf{Para}$



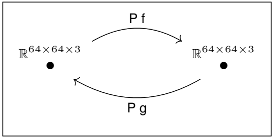
- Start with a functor $\mathbf{Free(G)} \to \mathbf{Para}$
  - Specify how it acts on objects



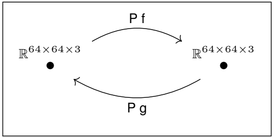
- Start with a functor $\mathbf{Free(G)} \to \mathbf{Para}$
  - Specify how it acts on objects
  - Start with randomly initialized morphisms



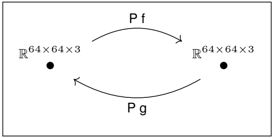
- Start with a functor $\mathbf{Free(G)} \to \mathbf{Para}$
  - Specify how it acts on objects
  - Start with randomly initialized morphisms
    - Every morphism in $\mathbf{Para}$ is a function parametrized by some P



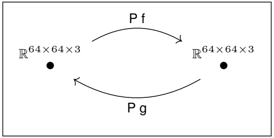
- Start with a functor $\mathbf{Free}(\mathbf{G}) \to \mathbf{Para}$
  - Specify how it acts on objects
  - Start with randomly initialized morphisms
    - Every morphism in $\mathbf{Para}$ is a function parametrized by some P
    - Initializing P randomly => “initializing” a morphism



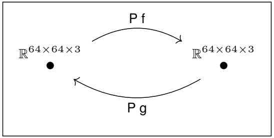
- Start with a functor $\mathbf{Free}(\mathbf{G}) \to \mathbf{Para}$
  - Specify how it acts on objects
  - Start with randomly initialized morphisms
    - Every morphism in $\mathbf{Para}$ is a function parametrized by some P
    - Initializing P randomly => “initializing” a morphism
- Get data samples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



- Start with a functor $\mathbf{Free(G)} \to \mathbf{Para}$
  - Specify how it acts on objects
  - Start with randomly initialized morphisms
    - Every morphism in $\mathbf{Para}$ 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 $\mathcal{C}$ and in every iteration:
  - For every morphism $(f : A \to B)$ in the transitive reduction of morphisms in $\mathcal{C}$, find Pf and minimize the distance between $(Pf)(d_a)$ and the corresponding image manifold



- Start with a functor $\mathbf{Free(G)} \to \mathbf{Para}$
  - Specify how it acts on objects
  - Start with randomly initialized morphisms
    - Every morphism in $\mathbf{Para}$ 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 $\mathcal{C}$ and in every iteration:
  - For every morphism ($f : A \to B$) in the transitive reduction of morphisms in $\mathcal{C}$, find Pf and minimize the distance between $(Pf)(d_a)$ and the corresponding image manifold
  - For all **path equations** from $A \to B$ 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 $\mathbf{Free}(\mathbf{G}) \to \mathbf{Para}$
  - Specify how it acts on objects
  - Start with randomly initialized morphisms
    - Every morphism in $\mathbf{Para}$ 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 $\mathcal{C}$ and in every iteration:
  - For every morphism $(f : A \to B)$ in the transitive reduction of morphisms in $\mathcal{C}$, find Pf and minimize the distance between $(Pf)(d_a)$ and the corresponding image manifold
  - For all **path equations** from $A \to B$ 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, $\mathcal{C}$

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

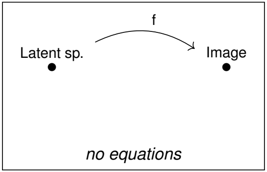


Figure: GAN

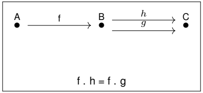


Figure: Equalizer

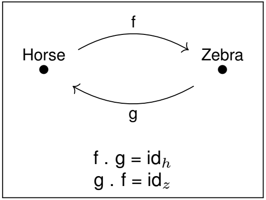


Figure: CycleGAN

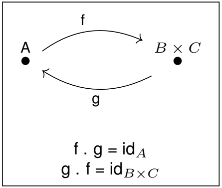
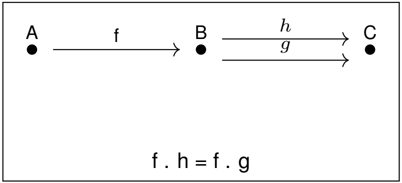


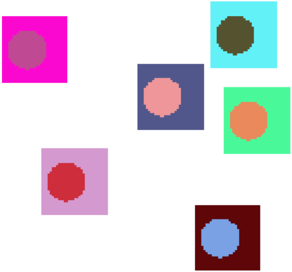
Figure: Product

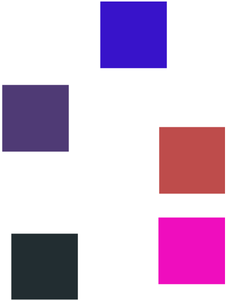
Equalizer schema



- Given two networks $h, g : B \to C$, find a subset $B' \subseteq B$ such that $B' = \{b \in B \mid h(b) = g(b)\}$

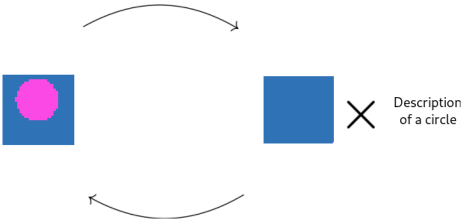
Consider two sets of images



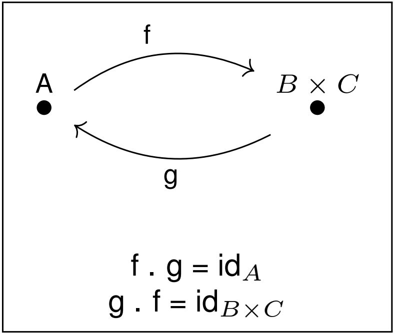


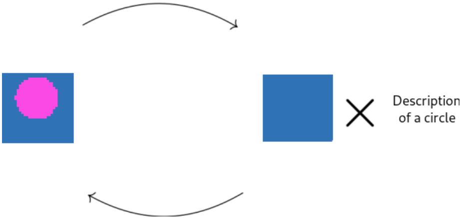
- Left: Background of color X with a circle with fixed size and position of color Y
- Right: Background of color Z

Product schema

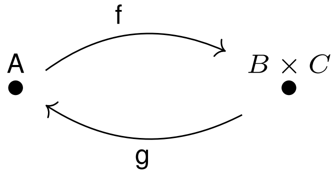


Product schema

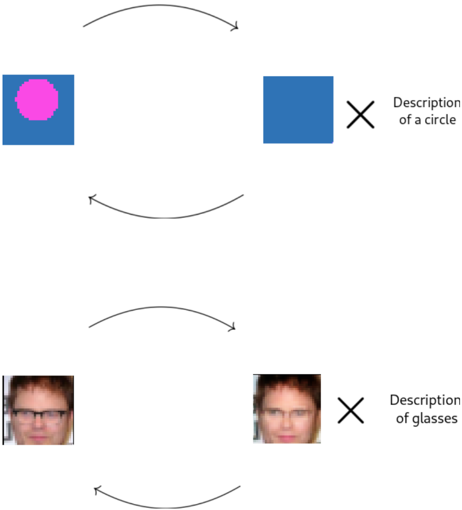




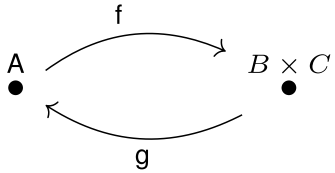
Product schema



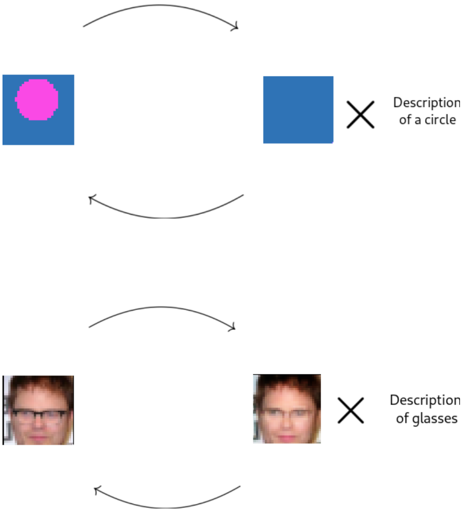
$$f \,.\, g = id_A$$
$$g \,.\, f = id_{B \times C}$$



Product schema



$$f \,.\, g = id_A$$
$$g \,.\, f = id_{B \times C}$$



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

Experiments

- CelebA dataset of 200K images of human faces

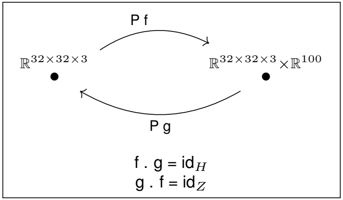
Experiments

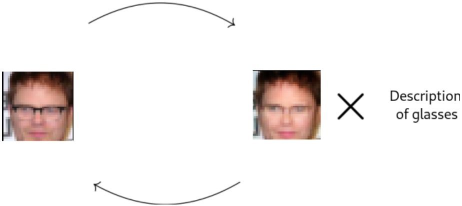
- CelebA dataset of 200K images of human faces

- Conveniently, there is a “glasses” annotation

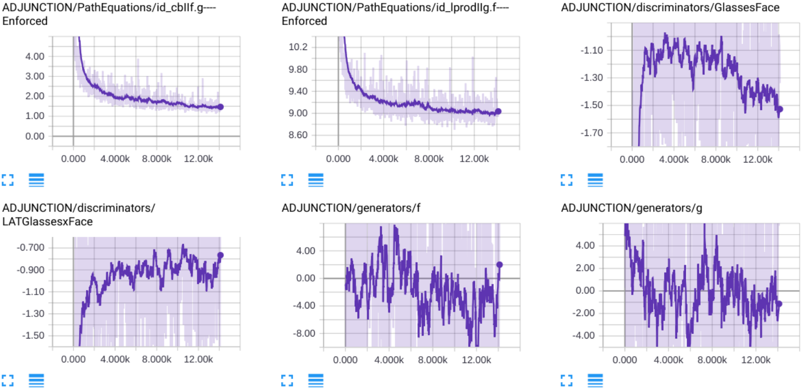
Experiments

$\mathrm{PC} :=$





- Collection of neural networks with total 40m parameters
- 7h training on a GeForce GTX 1080
- Successful results


ADJUNCTION/PathEquations/id_cbIIf.g----Enforced
4.00
3.00
2.00
1.00
0.00
0.000
4.000k
8.000k
12.00k
ADJUNCTION/PathEquations/id_lprodIIg.f----Enforced
10.2
9.80
9.40
9.00
8.60
0.000
4.000k
8.000k
12.00k
ADJUNCTION/discriminators/GlassesFace
-1.10
-1.30
-1.50
-1.70
0.000
4.000k
8.000k
12.00k
ADJUNCTION/discriminators/LATGlassesxFace
-0.700
-0.900
-1.10
-1.30
-1.50
0.000
4.000k
8.000k
12.00k
ADJUNCTION/generators/f
4.00
0.00
-4.00
-8.00
0.000
4.000k
8.000k
12.00k
ADJUNCTION/generators/g
4.00
2.00
0.00
-2.00
-4.00
0.000
4.000k
8.000k
12.00k

Experiments

Figure: Same image, different Z vector

Experiments

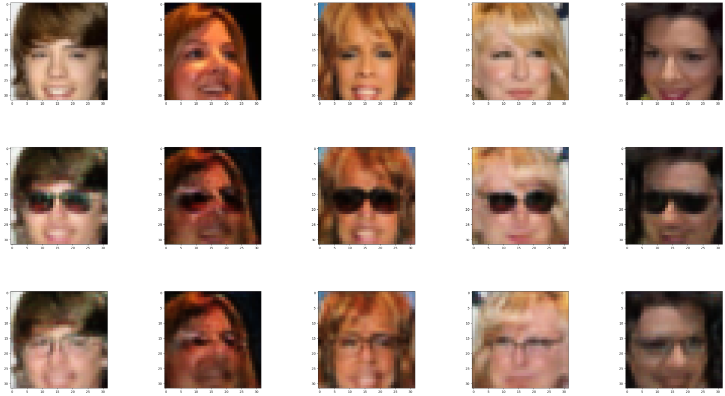


Figure: Same Z vector, different image

Experiments

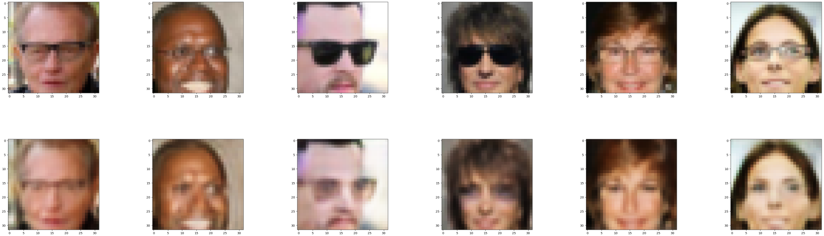


Figure: Top row: original image, bottom row: Removed glasses

Conclusions

- 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 informaton 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 informaton 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 informaton 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 informaton 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!