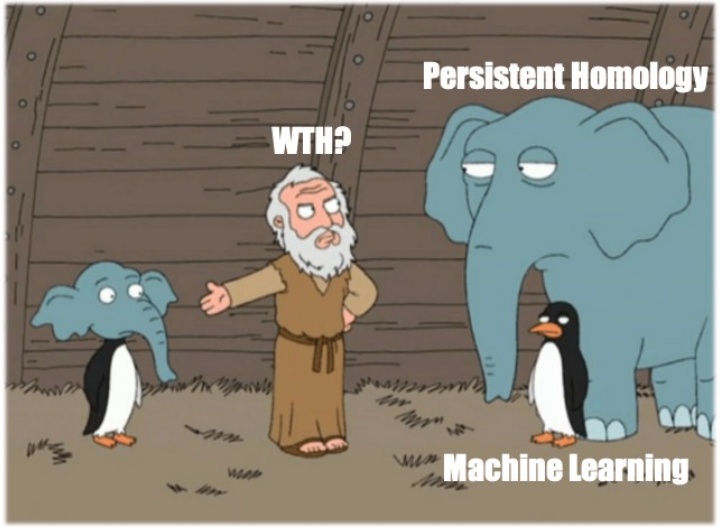


Note: All references marked are clickable!

Learning from and with persistent homology

Roland Kwitt

@rkwitt1982
http://rkwitt.org

Thematic Mini-Conference on Computational Topology & Machine Learning, September 2021

Talk outline

- Quick recap of the **learning framework** (supervised learning)
- Neural networks
- Learning **from** persistent homology
- Learning **with** persistent homology

Problem setting (of supervised learning)

Domain set	$\mathcal{X}$ (e.g., $\mathbb{R}^d$)
Label set	$\mathcal{Y}$ (e.g., $\{0, 1\}$)
Hypothesis class	$\mathcal{H}$
Distribution over domain & labels	$(x_i, y_i) \sim \mathcal{P}$
Training data	$S = ((x_1, y_1), \ldots, (x_m, y_m)) \sim \mathcal{P}^m$

Problem setting (of supervised learning)

Domain set	$\mathcal{X}$ (e.g., $\mathbb{R}^d$)
Label set	$\mathcal{Y}$ (e.g., $\{0, 1\}$)
Hypothesis class	$\mathcal{H}$
Distribution over domain & labels	$(x_i, y_i) \sim \mathcal{P}$
Training data	$S = ((x_1, y_1), \ldots, (x_m, y_m)) \sim \mathcal{P}^m$

A **learner** (upon receiving training data) needs to output a hypothesis

$$\mathcal{H} \ni h : \mathcal{X} \to \mathcal{Y}$$

Problem setting (of supervised learning)

Domain set	$\mathcal{X}$ (e.g., $\mathbb{R}^d$)
Label set	$\mathcal{Y}$ (e.g., $\{0, 1\}$)
Hypothesis class	$\mathcal{H}$
Distribution over domain & labels	$(x_i, y_i) \sim \mathcal{P}$
Training data	$S = ((x_1, y_1), \ldots, (x_m, y_m)) \sim \mathcal{P}^m$

A **learner** (upon receiving training data) needs to output a hypothesis

$$\mathcal{H} \ni h : \mathcal{X} \to \mathcal{Y}$$

Such a hypothesis should have **small risk**, defined as

$$\boxed{L_{\mathcal{P}}(h) = \mathbf{Pr}_{(x,y)\sim\mathcal{P}}[h(x) \neq y]}$$

Problem setting (of supervised learning)

However, we can only measure the **empirical risk**

$$L_S(h) = \frac{|i \in \{1, \ldots, m\} : h(x_i) \neq y_i|}{m}$$

Problem setting (of supervised learning)

However, we can only measure the **empirical risk**

$$L_S(h) = \frac{|i \in \{1, ..., m\} : h(x_i) \neq y_i|}{m}$$

Classic learning paradigm: minimize **empirical risk**

$$\boxed{h \in \arg\min_{h \in \mathcal{H}} L_S(h)}$$

Problem setting (of supervised learning)

However, we can only measure the **empirical risk**

$$L_S(h) = \frac{|i \in \{1,\dots,m\} : h(x_i) \neq y_i|}{m}$$

Classic learning paradigm: minimize **empirical risk**

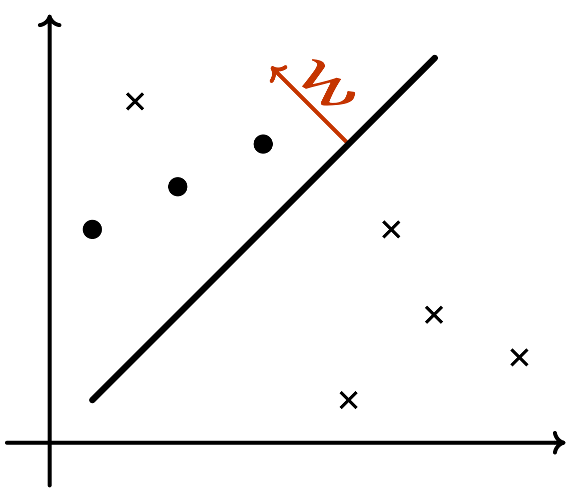
$$\boxed{h \in \arg\min_{h \in \mathcal{H}} L_S(h)}$$

Example:

$$\mathcal{X} = \mathbb{R}^d, \mathcal{Y} = \{+1, -1\}$$

$$\mathcal{H} = \{\boldsymbol{x} \mapsto \operatorname{sgn}\langle \boldsymbol{x}, \boldsymbol{w}\rangle : \boldsymbol{w} \in \mathbb{R}^d\}$$

(aka halfspace classifiers)



Problem setting (of supervised learning)

An important aspect is that, typically, inputs are of **fixed** size!

Problem setting (of supervised learning)

An important aspect is that, typically, inputs are of **fixed** size!

Other types of data, such as

- ▷ **sets**,
- ▷ **multi-sets**,
- ▷ **graphs**, or
- ▷ **point clouds**

are (or were) – lets put it this way – more challenging to handle!

Problem setting (of supervised learning)

An important aspect is that, typically, inputs are of **fixed** size!

Other types of data, such as

- **sets**,
- **multi-sets**,
- **graphs**, or
- **point clouds**

are (or were) – lets put it this way – more challenging to handle!

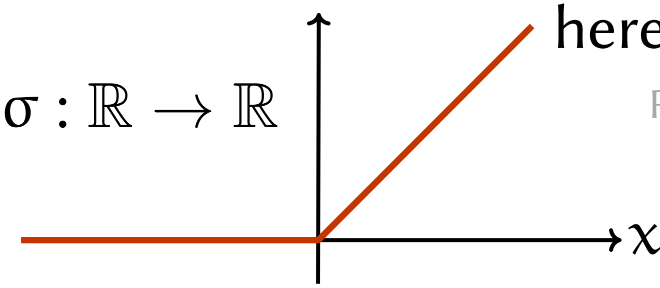
General recipe: Find a reasonable way to vectorize!

Typical (feed-forward) neural networks compose maps of the form

$$
\begin{aligned}
f : \mathbb{R}^d &\to \mathbb{R}^e \\
\boldsymbol{x} &\mapsto \sigma(\boldsymbol{A}\boldsymbol{x})
\end{aligned}
$$

i.e., a linear map $\boldsymbol{A}$, followed by a (component-wise) activation, e.g.,

$\sigma : \mathbb{R} \to \mathbb{R}$



here: $\sigma(x) = \max\{0, x\}$

Rectified Linear Unit (ReLU)

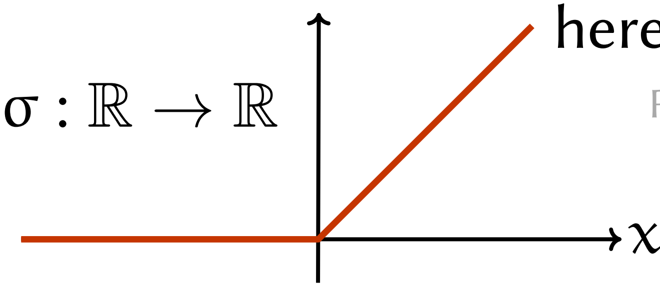
Neural networks

Typical (feed-forward) neural networks compose maps of the form

$$\begin{aligned} f : \mathbb{R}^d &\to \mathbb{R}^e \\ \boldsymbol{x} &\mapsto \sigma(\boldsymbol{A}\boldsymbol{x}) \end{aligned}$$

i.e., a linear map $\boldsymbol{A}$, followed by a (component-wise) activation, e.g.,

$\sigma : \mathbb{R} \to \mathbb{R}$



here: $\sigma(x) = \max\{0, x\}$

Rectified Linear Unit (ReLU)

Composition of such “building blocks” gives

$$\boxed{\begin{aligned} F : \mathbb{R}^d &\to \mathbb{R} \\ \boldsymbol{x} &\mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\boldsymbol{A}_1 \boldsymbol{x}) \cdots)) \end{aligned}}$$

i.e., the **hypothesis class** is parametrized by $(\boldsymbol{A}_1, \dots, \boldsymbol{A}_L, \boldsymbol{w})$.

Barcodes as input?

So, what if the input, $\boldsymbol{x}$, to

$$F : \mathbb{R}^d \to \mathbb{R}, \quad \boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\boldsymbol{A}_1 \boldsymbol{x}) \cdots))$$

Barcodes as input?

So, what if the input, $\boldsymbol{x}$, to

$$F : \mathbb{R}^d \to \mathbb{R}, \quad \boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\boldsymbol{A}_1 \boldsymbol{x}) \cdots))$$

is now a **persistence barcode**, G, i.e., a multi-set of (birth, death) tuple?

Barcodes as input?

So, what if the input, $\boldsymbol{x}$, to

$$F : \cancel{\mathbb{R}^d} \to \mathbb{R}, \quad \cancel{\boldsymbol{x}} \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\cancel{\boldsymbol{A}_1 \boldsymbol{x}}) \cdots))$$

is now a **persistence barcode**, G, i.e., a multi-set of (birth, death) tuple?

Barcodes as input?

So, what if the input, $\boldsymbol{x}$, to

$$F : \cancel{\mathbb{R}^d} \to \mathbb{R}, \quad \cancel{\boldsymbol{x}} \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\cancel{\boldsymbol{A}_1 \boldsymbol{x}}) \cdots))$$

is now a **persistence barcode**, G, i.e., a multi-set of (birth, death) tuple?

Question: How can we deal with this?

Barcodes as input?

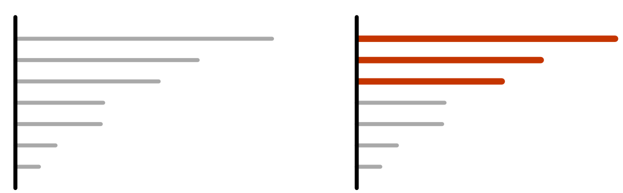
So, what if the input, $\boldsymbol{x}$, to

$$F : \cancel{\mathbb{R}^d} \to \mathbb{R}, \quad \cancel{\boldsymbol{x}} \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\cancel{\boldsymbol{A}_1 \boldsymbol{x}}) \cdots))$$

is now a **persistence barcode**, G, i.e., a multi-set of (birth, death) tuple?

Question: How can we deal with this?

A pragmatic approach [Bendich et al., 2014]:



take the **lengths** of the N-longest bars → gives a N-dim. vectorization

Barcodes as input?

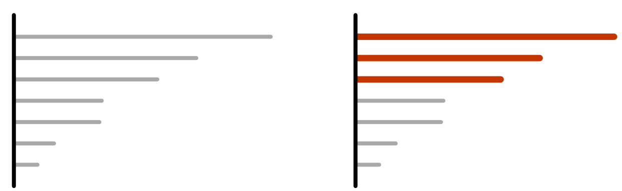
So, what if the input, $\boldsymbol{x}$, to

$$F : \cancel{\mathbb{R}^d} \to \mathbb{R}, \quad \cancel{\boldsymbol{x}} \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\cancel{\boldsymbol{A}_1} \boldsymbol{x}) \cdots))$$

is now a **persistence barcode**, G, i.e., a multi-set of (birth, death) tuple?

Question: How can we deal with this?

A pragmatic approach [Bendich et al., 2014]:



take the **lengths** of the N-longest bars → gives a N-dim. vectorization

Question: Why should we care about "how" we vectorize?

Well, it would be desirable to preserve **stability** wrt. d_B, $d_{W_{p,q}}$.

Prior art

Vectorization techniques

Persistence landscapes [Bubenik, 2015]
Persistence silhouettes [Chazal et al., 2014]
Persistence images [Adams et al., 2017]
Template functions [Perea et al., 2019]
ATOL† [Royer et al., 2019]

† actually an (unsupervised) learning technique

Prior art

Vectorization techniques

- Persistence landscapes [Bubenik, 2015]
- Persistence silhouettes [Chazal et al., 2014]
- Persistence images [Adams et al., 2017]
- Template functions [Perea et al., 2019]
- ATOL† [Royer et al., 2019]

Kernel-based techniques

- Persistence scale-space kernel [Reininghaus et al., 2014]
- Sliced Wasserstein kernel [Carrière et al., 2017]
- Persistence-weighted Gaussian kernel [Kusano et al., 2016]
- Kernel for multi-parameter persistent homology [Corbet et al., 2019]
- Theoretical results related to metric distortion [Carrière & Bauer, 2019]

† actually an (unsupervised) learning technique

Prior art

Vectorization techniques

- Persistence landscapes — [Bubenik, 2015]
- Persistence silhouettes — [Chazal et al., 2014]
- Persistence images — [Adams et al., 2017]
- Template functions — [Perea et al., 2019]
- ATOL† — [Royer et al., 2019]

Kernel-based techniques

- Persistence scale-space kernel — [Reininghaus et al., 2014]
- Sliced Wasserstein kernel — [Carrière et al., 2017]
- Persistence-weighted Gaussian kernel — [Kusano et al., 2016]
- Kernel for multi-parameter persistent homology — [Corbet et al., 2019]
- Theoretical results related to metric distortion — [Carrière & Bauer, 2019]

This is, by far, **not** an exhaustive listing!

† actually an (unsupervised) learning technique

Can we obtain task-optimal vectorizations?

In fact, most vectorization strategies are **task-agnostic**!

Question: Shouldn't the vectorization be informed by the learning task?

Can we obtain task-optimal vectorizations?

In fact, most vectorization strategies are **task-agnostic**!

Question: Shouldn't the vectorization be informed by the learning task?

This motivates **learnable** vectorization schemes:

[Hofer et al., 2017,2019], [Carrière et al., 2019]

Can we obtain task-optimal vectorizations?

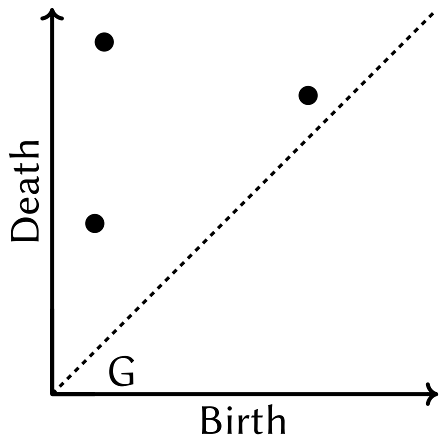
In fact, most vectorization strategies are **task-agnostic**!

Question: Shouldn't the vectorization be informed by the learning task?

This motivates **learnable** vectorization schemes:

[Hofer et al., 2017,2019], [Carrière et al., 2019]

Example (for a vectorization into $\mathbb{R}^k, k = 2$):



Can we obtain task-optimal vectorizations?

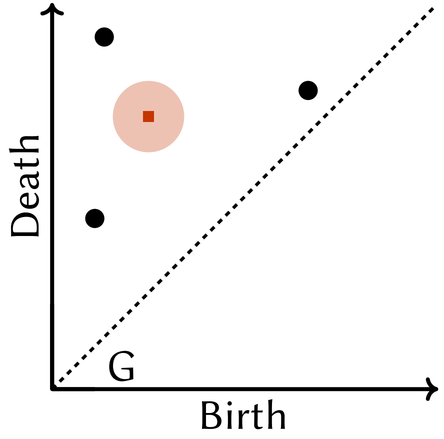
In fact, most vectorization strategies are **task-agnostic**!

Question: Shouldn't the vectorization be informed by the learning task?

This motivates **learnable** vectorization schemes:

[Hofer et al., 2017,2019], [Carrière et al., 2019]

Example (for a vectorization into $\mathbb{R}^k$, $k = 2$):



Can we obtain task-optimal vectorizations?

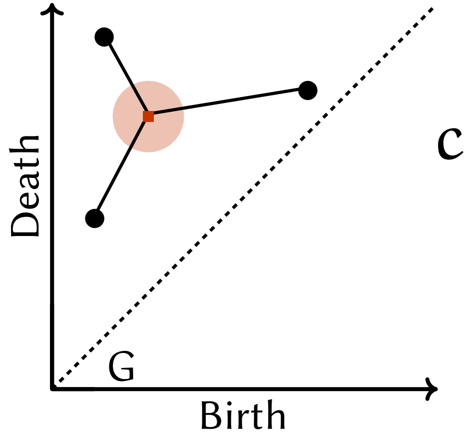
In fact, most vectorization strategies are **task-agnostic**!

Question: Shouldn't the vectorization be informed by the learning task?

This motivates **learnable** vectorization schemes:

[Hofer et al., 2017,2019], [Carrière et al., 2019]

Example (for a vectorization into $\mathbb{R}^k, k = 2$):



$$c_1 = \sum_{p \in G} s_{\theta_1}(p)$$

Can we obtain task-optimal vectorizations?

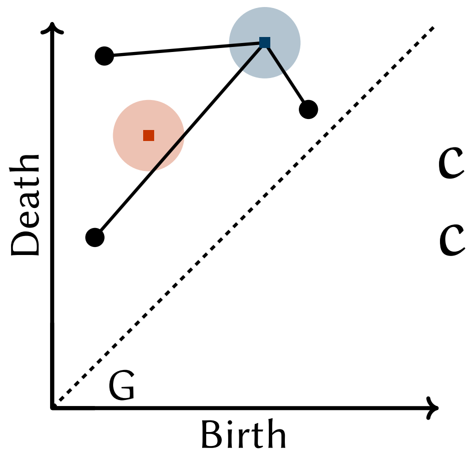
In fact, most vectorization strategies are **task-agnostic**!

Question: Shouldn't the vectorization be informed by the learning task?

This motivates **learnable** vectorization schemes:

[Hofer et al., 2017,2019], [Carrière et al., 2019]

Example (for a vectorization into $\mathbb{R}^k$, $k = 2$):



$$c_1 = \sum_{p \in G} s_{\theta_1}(p)$$
$$c_2 = \sum_{p \in G} s_{\theta_2}(p)$$

Can we obtain task-optimal vectorizations?

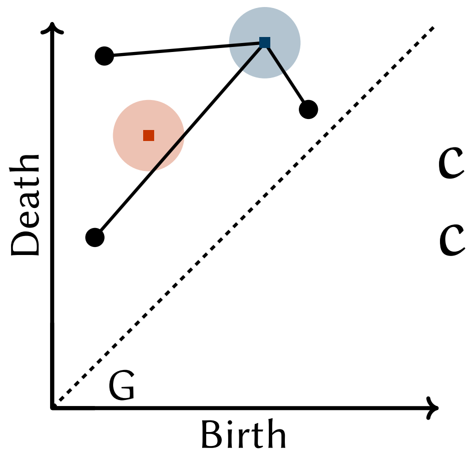
In fact, most vectorization strategies are **task-agnostic**!

Question: Shouldn't the vectorization be informed by the learning task?

This motivates **learnable** vectorization schemes:

[Hofer et al., 2017,2019], [Carrière et al., 2019]

Example (for a vectorization into $\mathbb{R}^k$, $k = 2$):



$$\left.\begin{aligned} c_1 &= \textstyle\sum_{p\in G} s_{\theta_1}(p) \\ c_2 &= \textstyle\sum_{p\in G} s_{\theta_2}(p) \end{aligned}\right\} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

In general†: $G \mapsto \mathcal{V}_\Theta(G)$

$\Theta = (\theta_1, \theta_2)$

† plus some technicalities to ensure stability

Can we obtain task-optimal vectorizations?

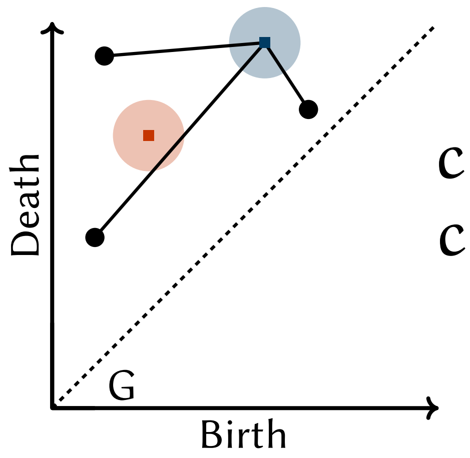
In fact, most vectorization strategies are **task-agnostic**!

Question: Shouldn't the vectorization be informed by the learning task?

This motivates **learnable** vectorization schemes:

[Hofer et al., 2017,2019], [Carrière et al., 2019]

Example (for a vectorization into $\mathbb{R}^k$, $k = 2$):



$$\left.\begin{aligned} c_1 &= \textstyle\sum_{p \in G} s_{\theta_1}(p) \\ c_2 &= \textstyle\sum_{p \in G} s_{\theta_2}(p) \end{aligned}\right\} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

In general†: $G \mapsto \mathcal{V}_\Theta(G)$

$\Theta = (\theta_1, \theta_2)$

† plus some technicalities to ensure stability

Learnable means that we can optimize the θ_i's for a given task/criterion!

Can we obtain task-optimal vectorizations?

Overall, this changes

$$F : \mathbb{R}^d \to \mathbb{R}, \quad \boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\boldsymbol{A}_1 \boldsymbol{x}) \cdots))$$

to

$$F : \mathcal{B} \to \mathbb{R}, \quad G \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\boldsymbol{A}_1 \mathcal{V}_\Theta(G)) \cdots))$$

† and, obviously for all $\boldsymbol{A}_i$'s as well

Can we obtain task-optimal vectorizations?

Overall, this changes

$$F : \mathbb{R}^d \to \mathbb{R}, \quad \boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\boldsymbol{A}_1 \boldsymbol{x}) \cdots))$$

to

$$F : \mathcal{B} \to \mathbb{R}, \quad G \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\boldsymbol{A}_1 \mathcal{V}_\Theta(G)) \cdots))$$

Upon the definition of a suitable **loss function**

$$\ell : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$$

we can compute, for a training sample, (G_i, y_i), the **parameter update**†

$$\boxed{\Theta^{t+1} = \Theta^t - \eta \frac{\partial \ell(F, (G_i, y_i))}{\partial \Theta}}$$

† and, obviously for all $\boldsymbol{A}_i$'s as well

Can we obtain task-optimal vectorizations?

Overall, this changes

$$F : \mathbb{R}^d \to \mathbb{R}, \quad \boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\boldsymbol{A}_1 \boldsymbol{x}) \cdots))$$

to

$$F : \mathcal{B} \to \mathbb{R}, \quad G \mapsto \boldsymbol{w}^\top \sigma(\boldsymbol{A}_L \sigma(\boldsymbol{A}_{L-1} \cdots \sigma(\boldsymbol{A}_1 \mathcal{V}_\Theta(G)) \cdots))$$

Upon the definition of a suitable **loss function**

$$\ell : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$$

we can compute, for a training sample, (G_i, y_i), the **parameter update**†

$$\boxed{\Theta^{t+1} = \Theta^t - \eta \frac{\partial \ell(F, (G_i, y_i))}{\partial \Theta}}$$

"Easy" because of **automatic differentiaton** (e.g., using PyTorch).

† and, obviously for all $\boldsymbol{A}_i$'s as well

Transitioning to learning with PH

In ML, we have, for long, degraded PH to a “fancy” **feature extractor**.

Transitioning to learning with PH

In ML, we have, for long, degraded PH to a “fancy” **feature extractor**.

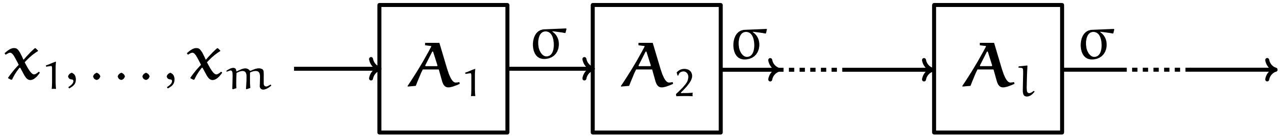
Question: What if we want to **control** topological properties?

Transitioning to learning with PH

In ML, we have, for long, degraded PH to a "fancy" **feature extractor**.

Question: What if we want to **control** topological properties?

Example:

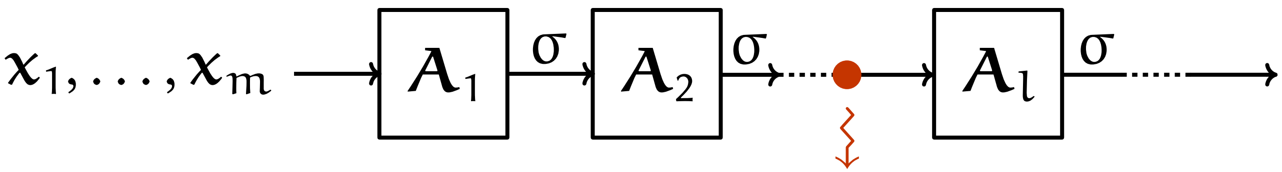


Transitioning to learning with PH

In ML, we have, for long, degraded PH to a "fancy" **feature extractor**.

Question: What if we want to **control** topological properties?

Example:

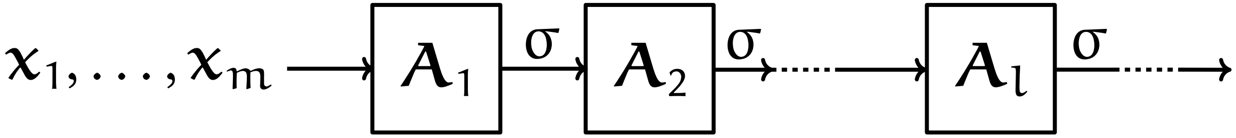


$z_1, \ldots, z_m$, with $z_i \in \mathbb{R}^h$

e.g., control the **lifetime** of 0-dim. features (from Vietoris-Rips PH)

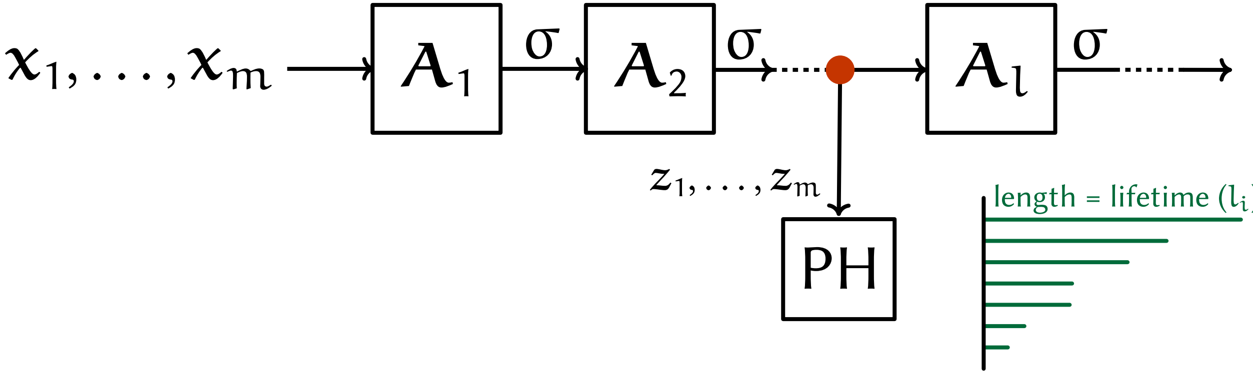
Transitioning to learning with PH

Example (contd.):



Transitioning to learning with PH

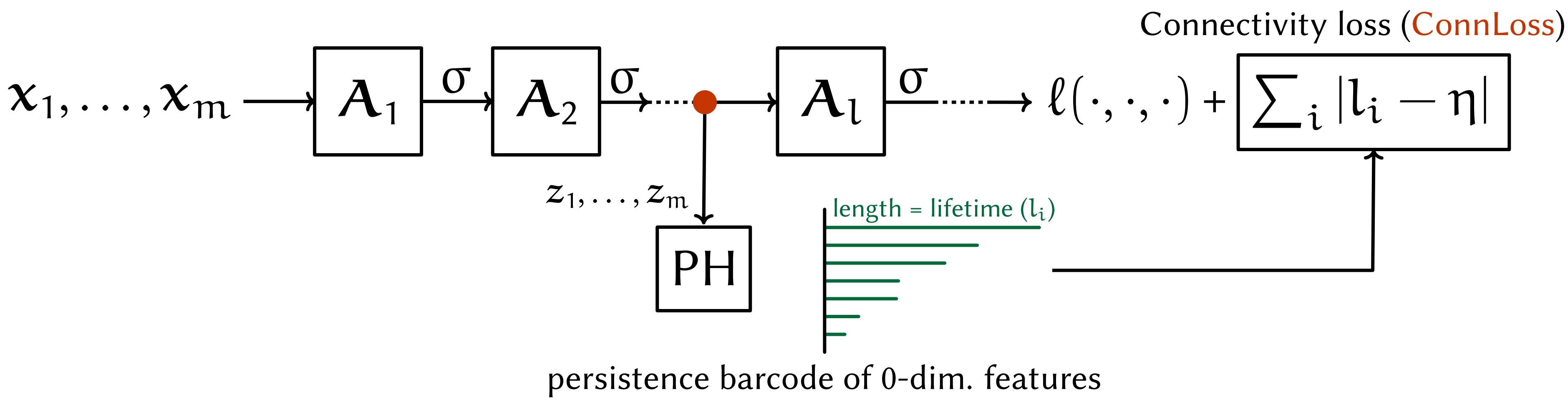
Example (contd.):



persistence barcode of 0-dim. features

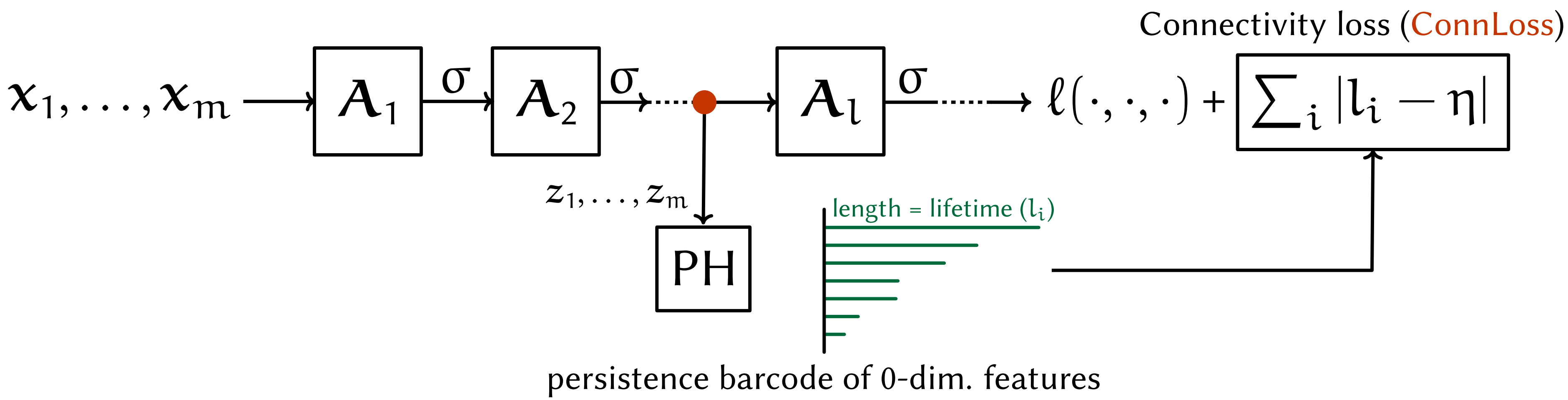
Transitioning to learning with PH

Example (contd.):



Transitioning to learning with PH

Example (contd.):

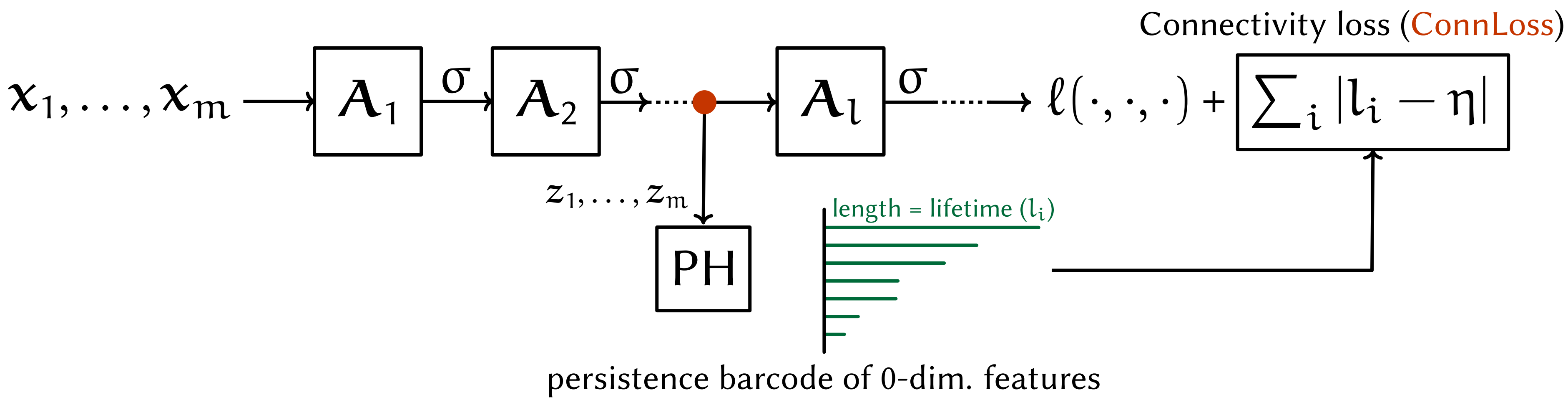


Importantly,

- ▷ the l_i's depend on the A_i's (as they influence the z_i's)

Transitioning to learning with PH

Example (contd.):

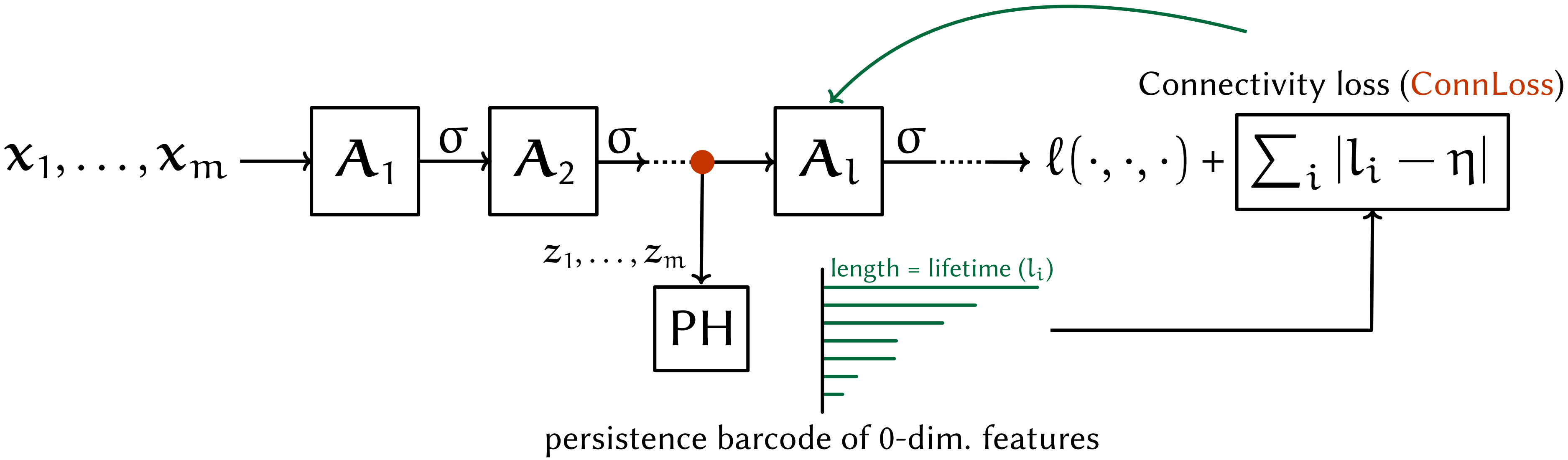


Importantly,

- the l_i's depend on the A_i's (as they influence the z_i's)
- **minimizing** the (joint) loss, requires gradients wrt. all A_i's

Transitioning to learning with PH

Example (contd.):

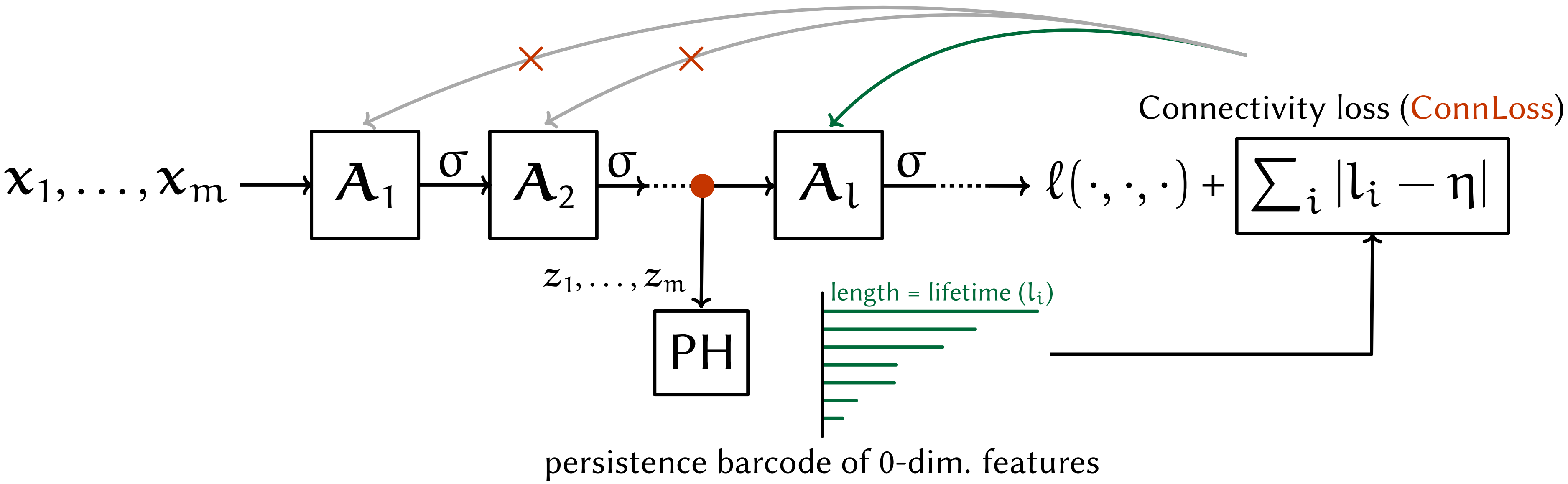


Importantly,

- the l_i's depend on the A_i's (as they influence the z_i's)
- **minimizing** the (joint) loss, requires gradients wrt. all A_i's

Transitioning to learning with PH

Example (contd.):

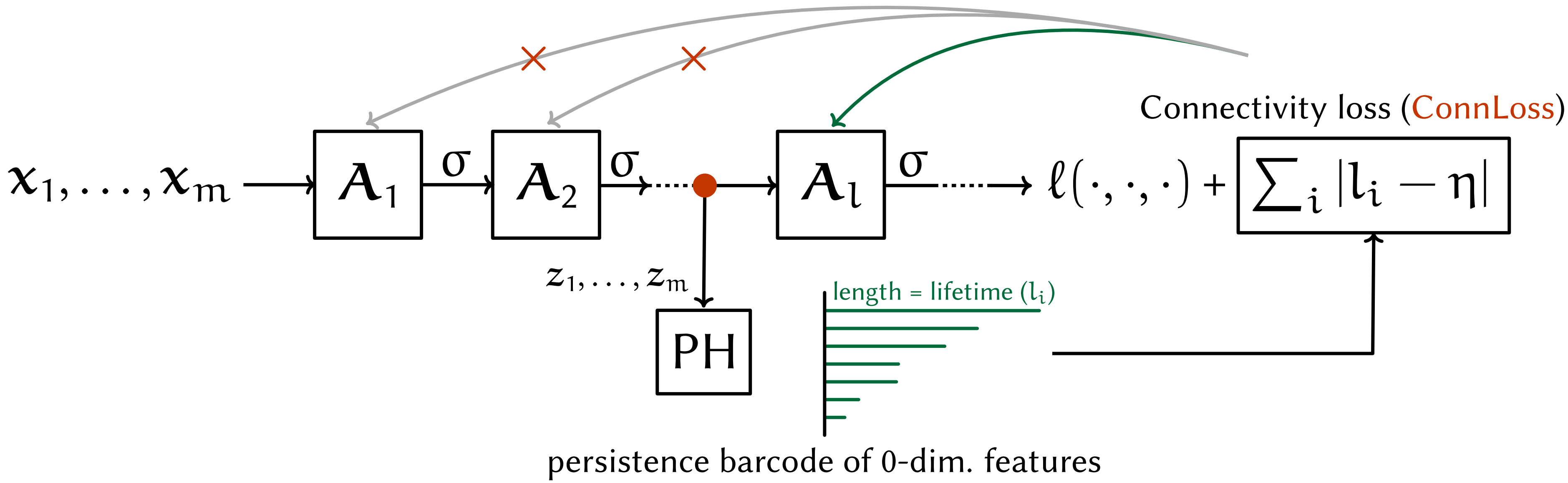


Importantly,

- ▷ the l_i's depend on the A_i's (as they influence the z_i's)
- ▷ **minimizing** the (joint) loss, requires gradients wrt. all A_i's

Transitioning to learning with PH

Example (contd.):



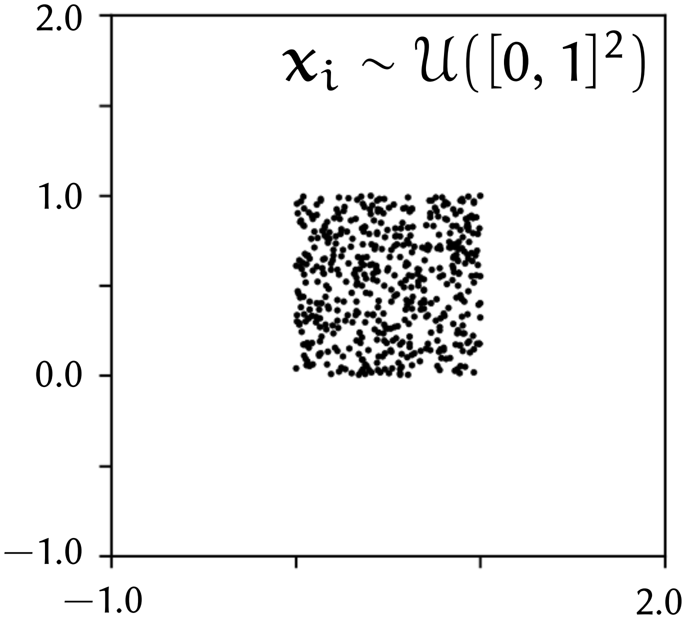
Importantly,

- ▷ the l_i's depend on the A_i's (as they influence the z_i's)
- ▷ **minimizing** the (joint) loss, requires gradients wrt. all A_i's
- ▷ The **good news** is that this can be done

[Hofer et al., 2019] [Carrière et al., 2020]
[Brüel-Gabrielsson et al., 2019]

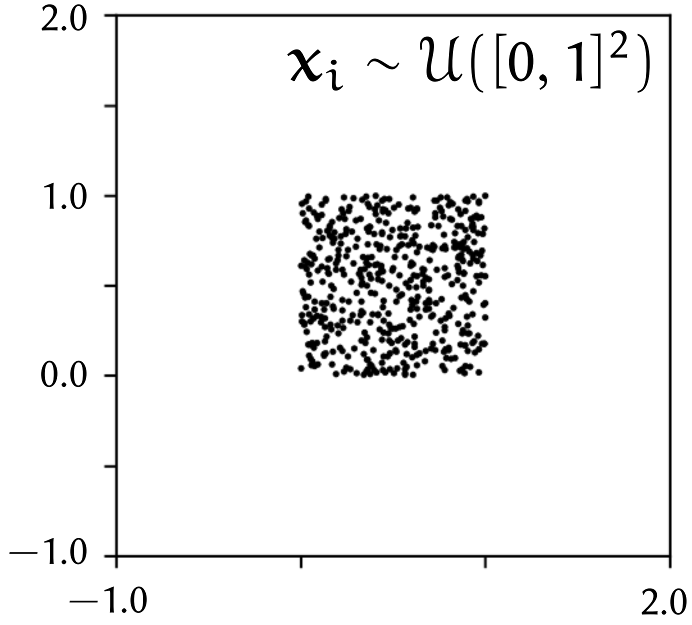
Transitioning to learning with PH

Lets look at some **toy data** first.



Transitioning to learning with PH

Lets look at some **toy data** first.

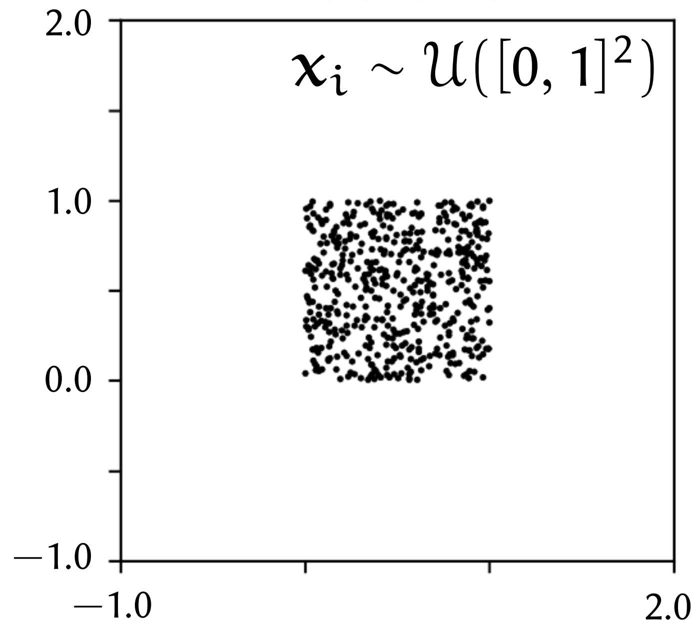


Here's what we aim to do:

- ▷ Compute 0-dim. Vietoris-Rips PH
- ▷ Minimize ConnLoss wrt. the x_i (for a desired $\eta > 0$)

Transitioning to learning with PH

Lets look at some **toy data** first.



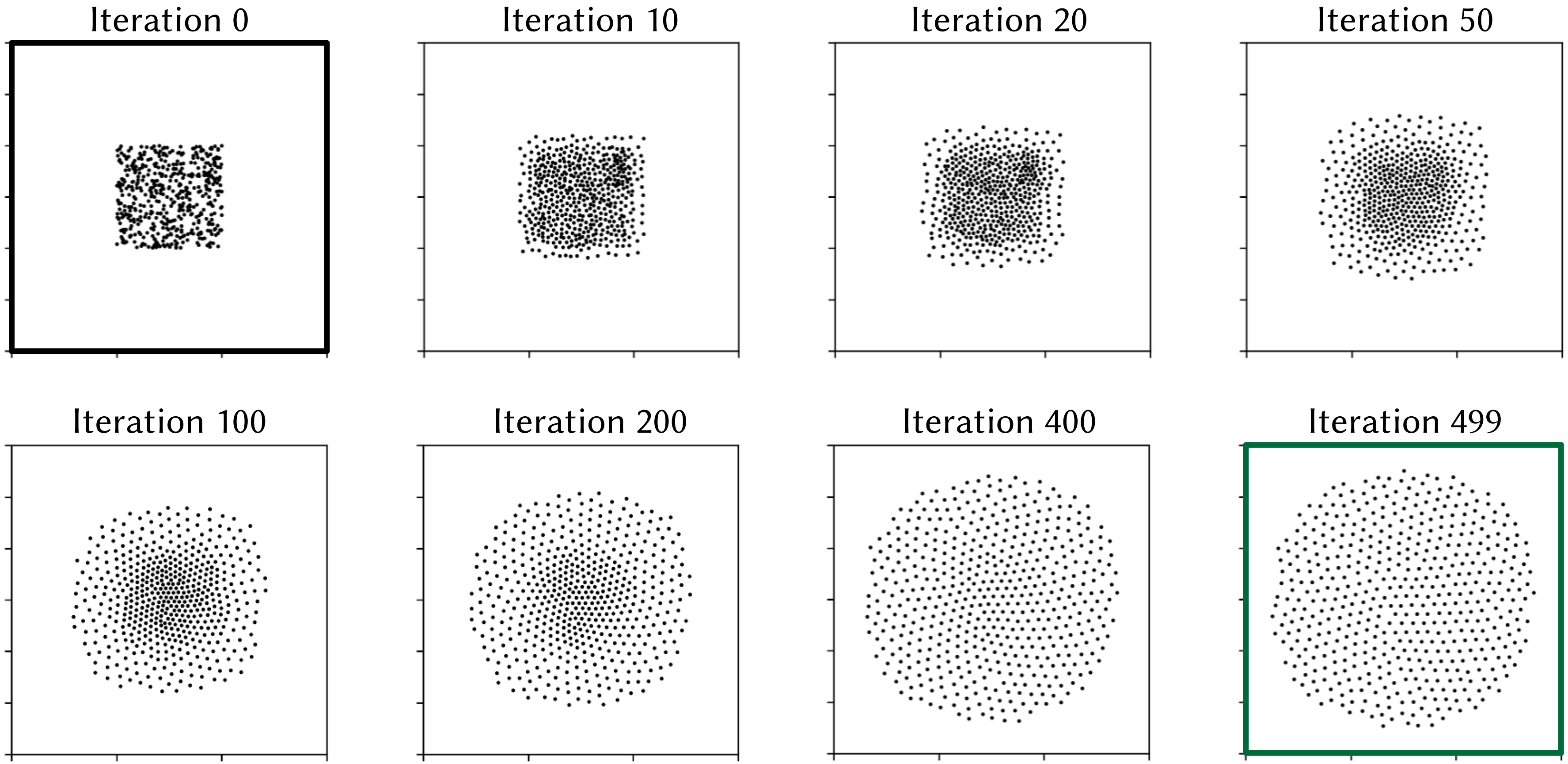
Here's what we aim to do:

- ▷ Compute 0-dim. Vietoris-Rips PH
- ▷ Minimize ConnLoss wrt. the x_i (for a desired $\eta > 0$)

Notably, this controls the **length** of the minimal spanning tree (MST).

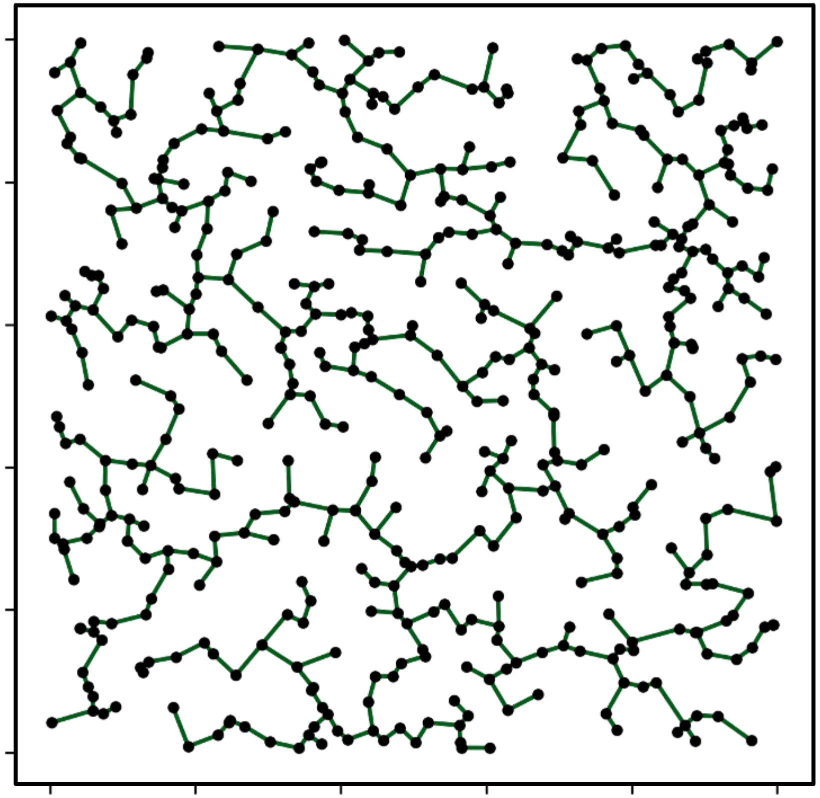
[Robins, 2000]

Transitioning to learning with PH

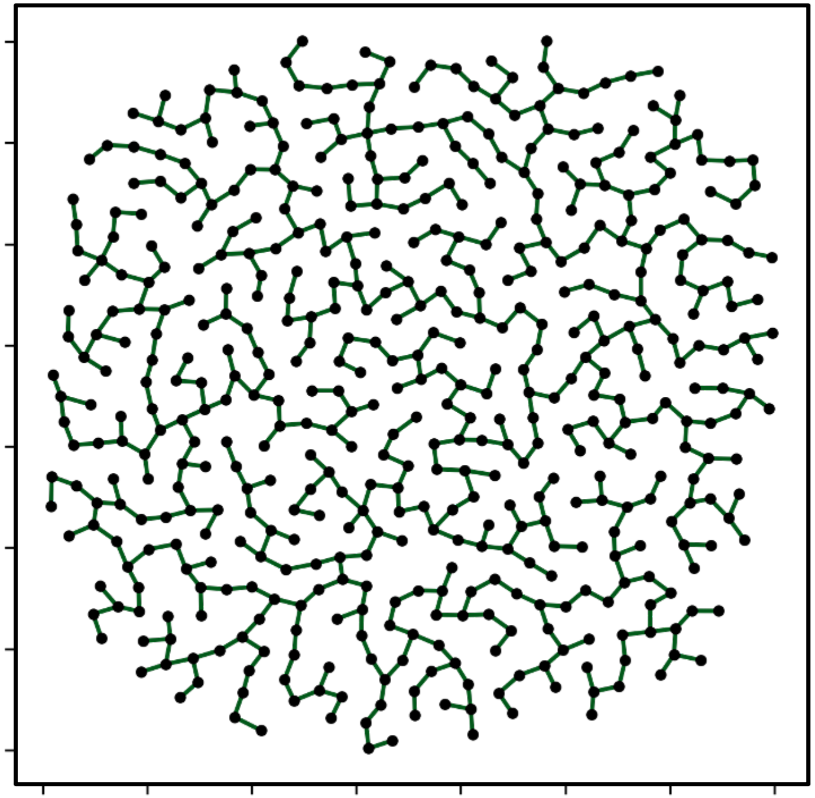


Transitioning to learning **with** PH

MST (Original)



MST (after optimization)



Some self-advertisement :)

Embedding into the PyTorch framework:

```
import torch
import numpy as np
from torchph.pershom import vr_persistence_l1

device = "cuda"

toy_data = np.random.rand(300, 2)
X = torch.tensor(toy_data, device=device, requires_grad=True)

opt = torch.optim.Adam([X], lr=0.01)

for i in range(1,100+1):
    pers = vr_persistence_l1(X, 1, 0)
    h_0 = pers[0][0]

    lt = h_0[:, 1] # H0 lifetimes
    loss = (lt - 0.1).abs().sum()

    opt.zero_grad()
    loss.backward()
    opt.step()
```

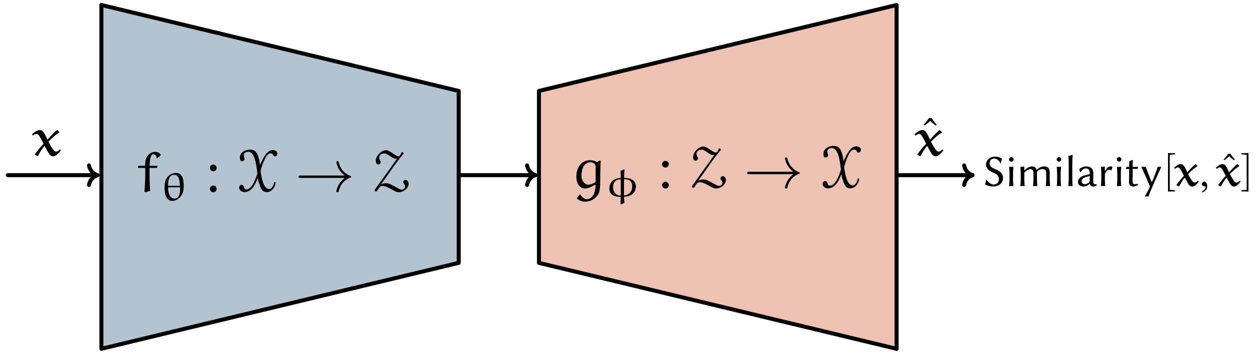
Note that this uses our own PH implementation (works on GPU), see

Why would this be useful?

In [Hofer et al., 2019], we study ConnLoss with **autoencoders**.

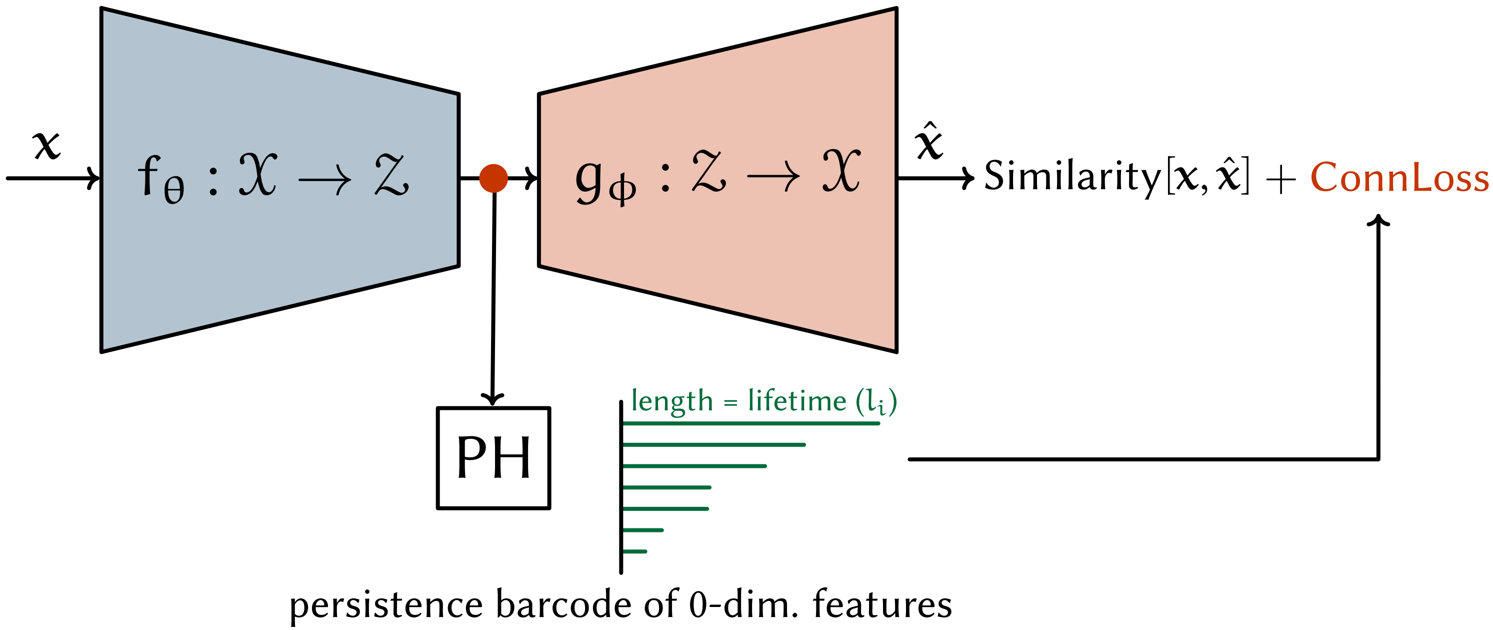
Why would this be useful?

In [Hofer et al., 2019], we study ConnLoss with **autoencoders**.



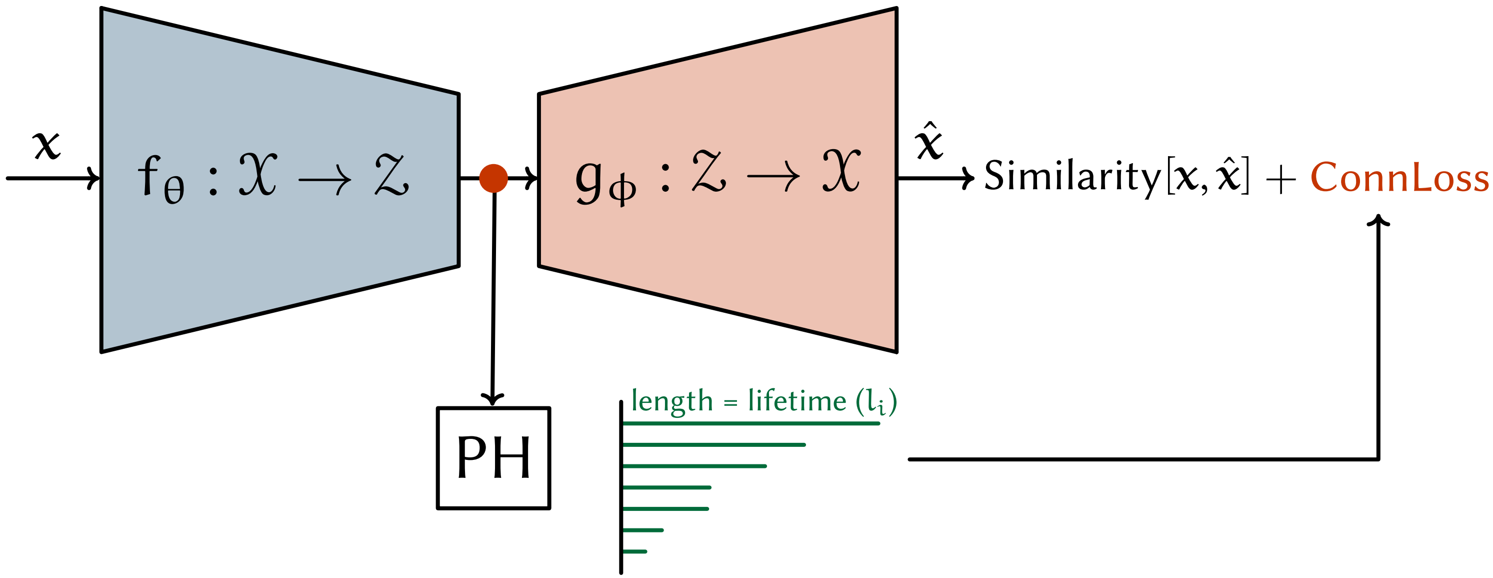
Why would this be useful?

In [Hofer et al., 2019], we study ConnLoss with **autoencoders**.



Why would this be useful?

In [Hofer et al., 2019], we study ConnLoss with **autoencoders**.

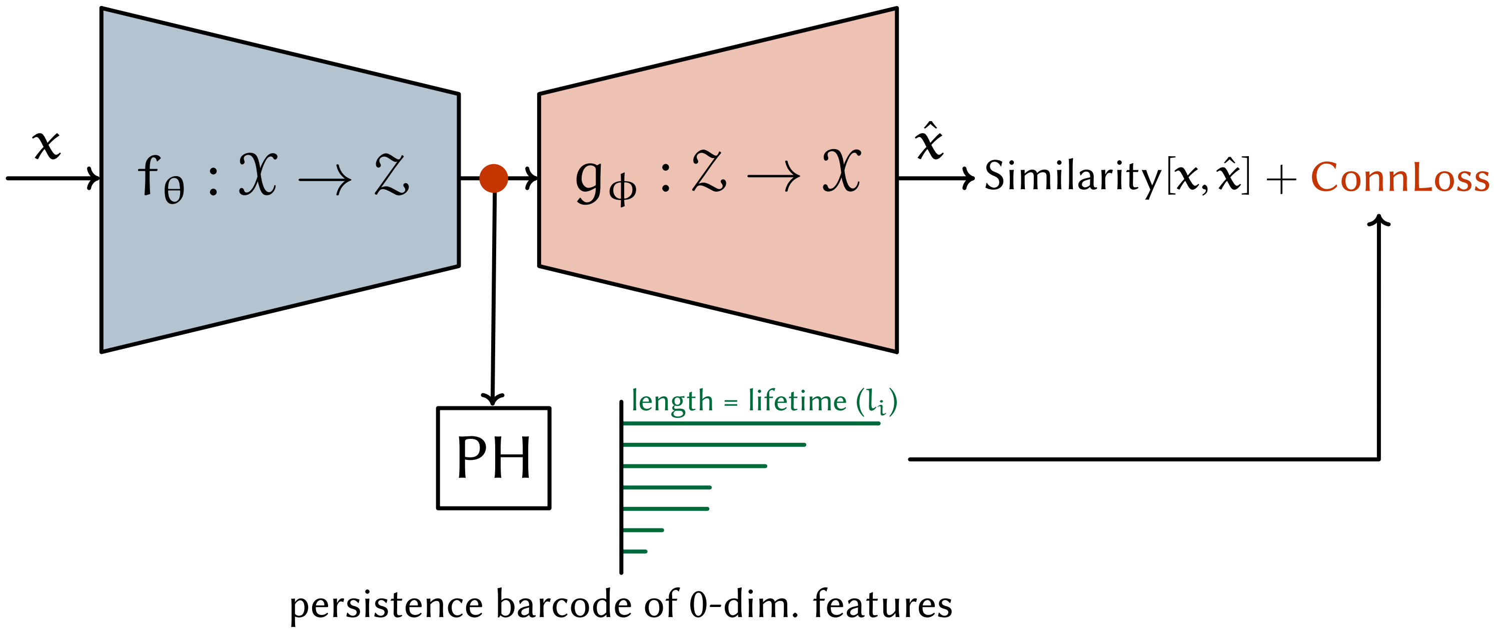


persistence barcode of 0-dim. features

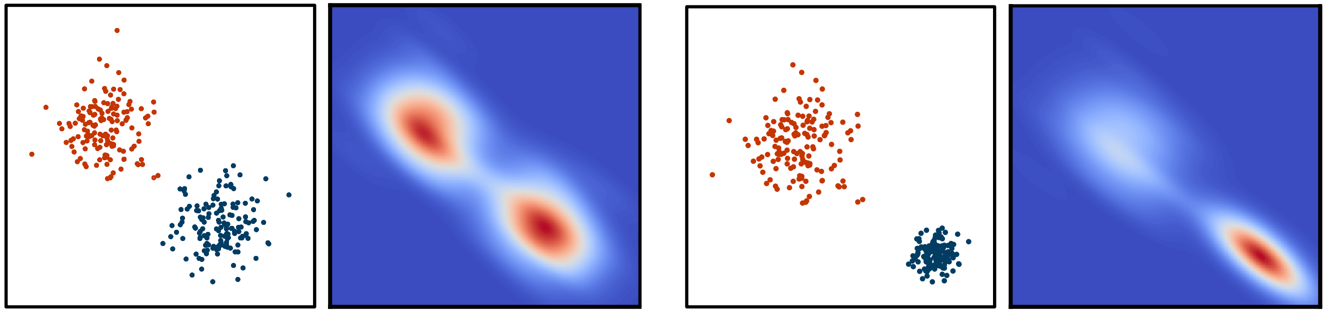
Why? You might want to do kernel density estimation in $\mathcal{Z}(= \mathbb{R}^n)$

Why would this be useful?

In [Hofer et al., 2019], we study ConnLoss with **autoencoders**.



Why? You might want to do kernel density estimation in $\mathcal{Z}(= \mathbb{R}^n)$

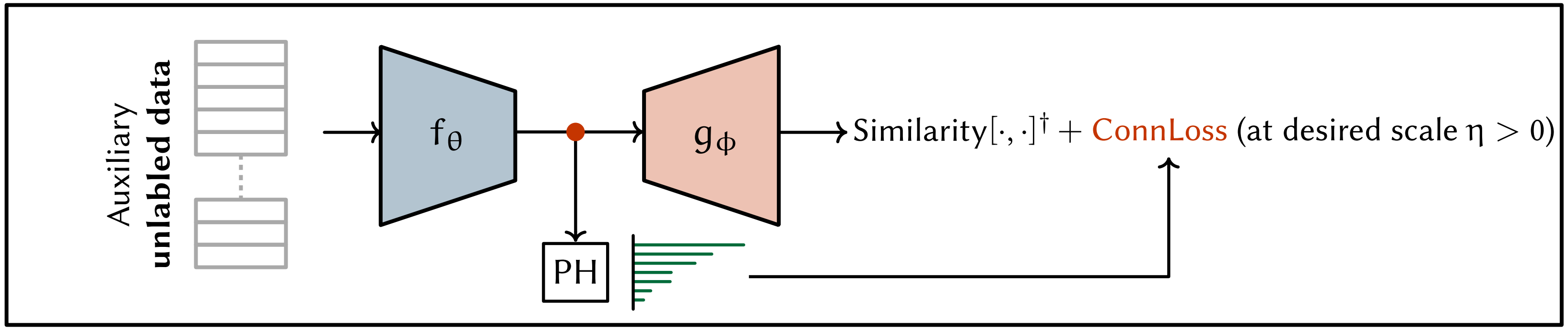


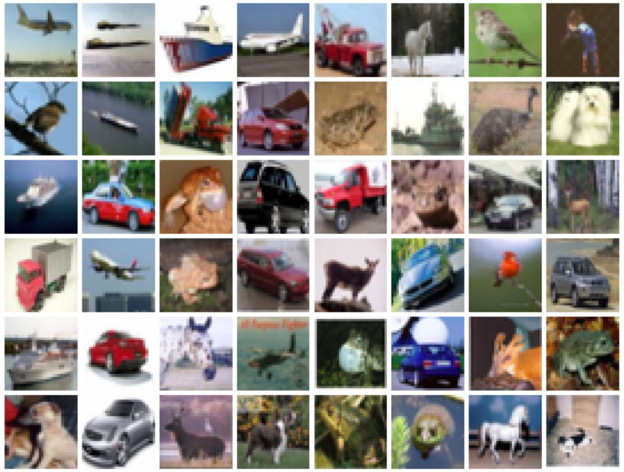
Can be problematic, due to scale differences → we can **impose** scale via η

Application: One-class learning

Training (step I)

Trained only **once** using unlabeled data





CIFAR10 images (32×32 RGB)

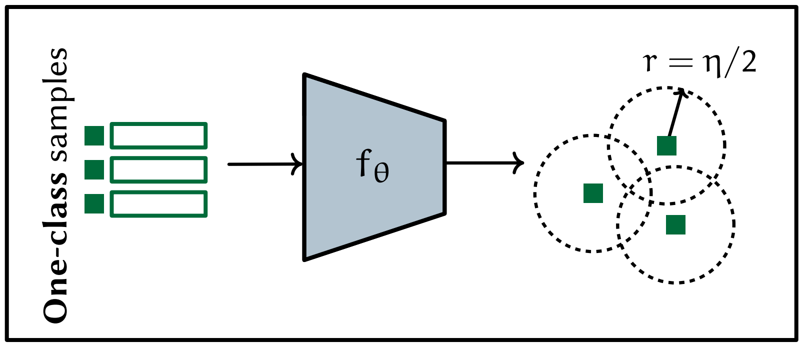
Notably, [Moor et al., 2019] follow similar ideas to learn a representation space ($\mathcal{Z}$) that preserves the input space topology.

† e.g.,Similarity$[\cdot,\cdot] \equiv$ mean squared-error (MSE)

Application: One-class learning

Training (step II)

KDE-inspired **one-class** "learning"

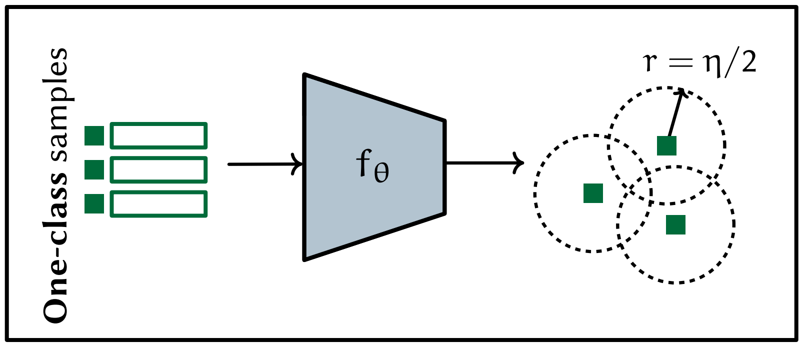


e.g., class = **dogs**

Application: One-class learning

Training (step II)

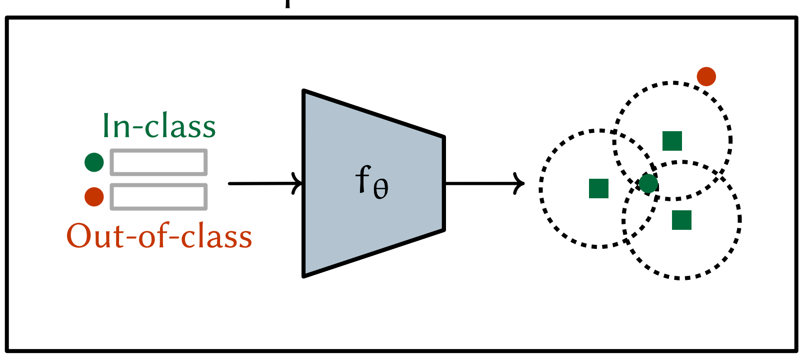
KDE-inspired **one-class** "learning"



e.g., class = **dogs**

Evaluation protocol

Computation of a **one-class** score



Count #samples falling into balls of radius $\eta/2$, anchored at the one-class instances ■

How about **neural classifiers**? [Hofer et al., 2020]

Application: Topological regularizers

How about **neural classifiers**? [Hofer et al., 2020]

Question: Can we **control** topological properties for generalization

Application: Topological regularizers

How about **neural classifiers**? [Hofer et al., 2020]

Question: Can we **control** topological properties for generalization

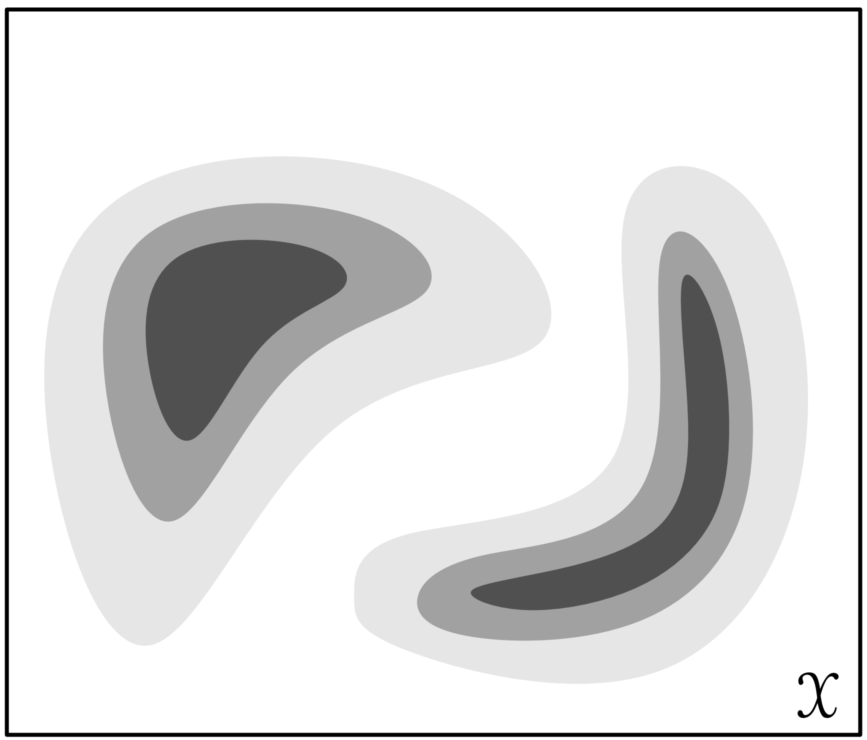
Key idea: encourage "densification" of learned representations

Application: Topological regularizers

How about **neural classifiers**? [Hofer et al., 2020]

Question: Can we **control** topological properties for generalization

Key idea: encourage "densification" of learned representations

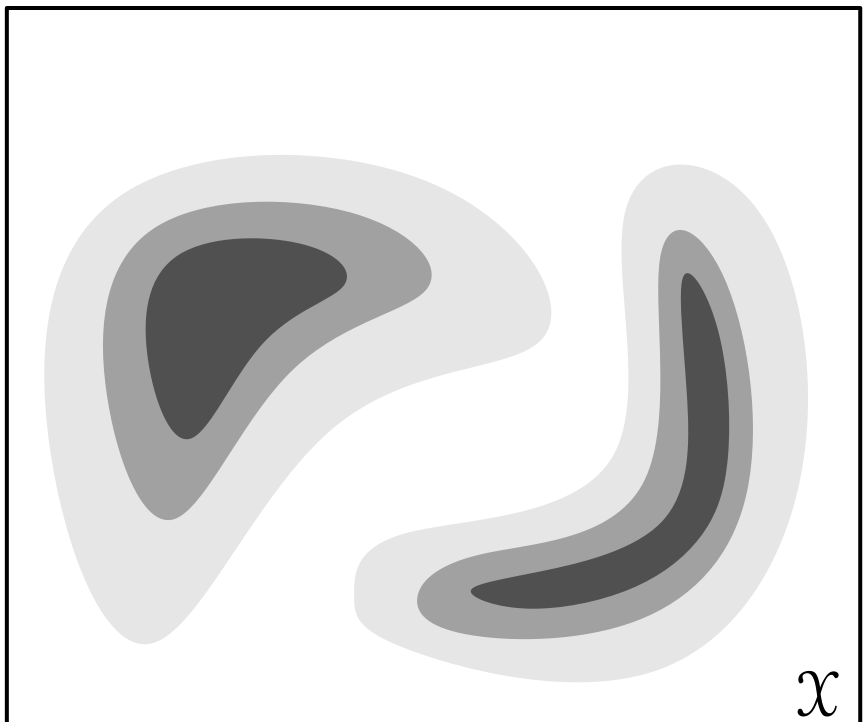


Application: Topological regularizers

How about **neural classifiers**? [Hofer et al., 2020]

Question: Can we **control** topological properties for generalization

Key idea: encourage "densification" of learned representations



$$\xrightarrow{c^\dagger} \{0, 1\} = \mathcal{Y}$$

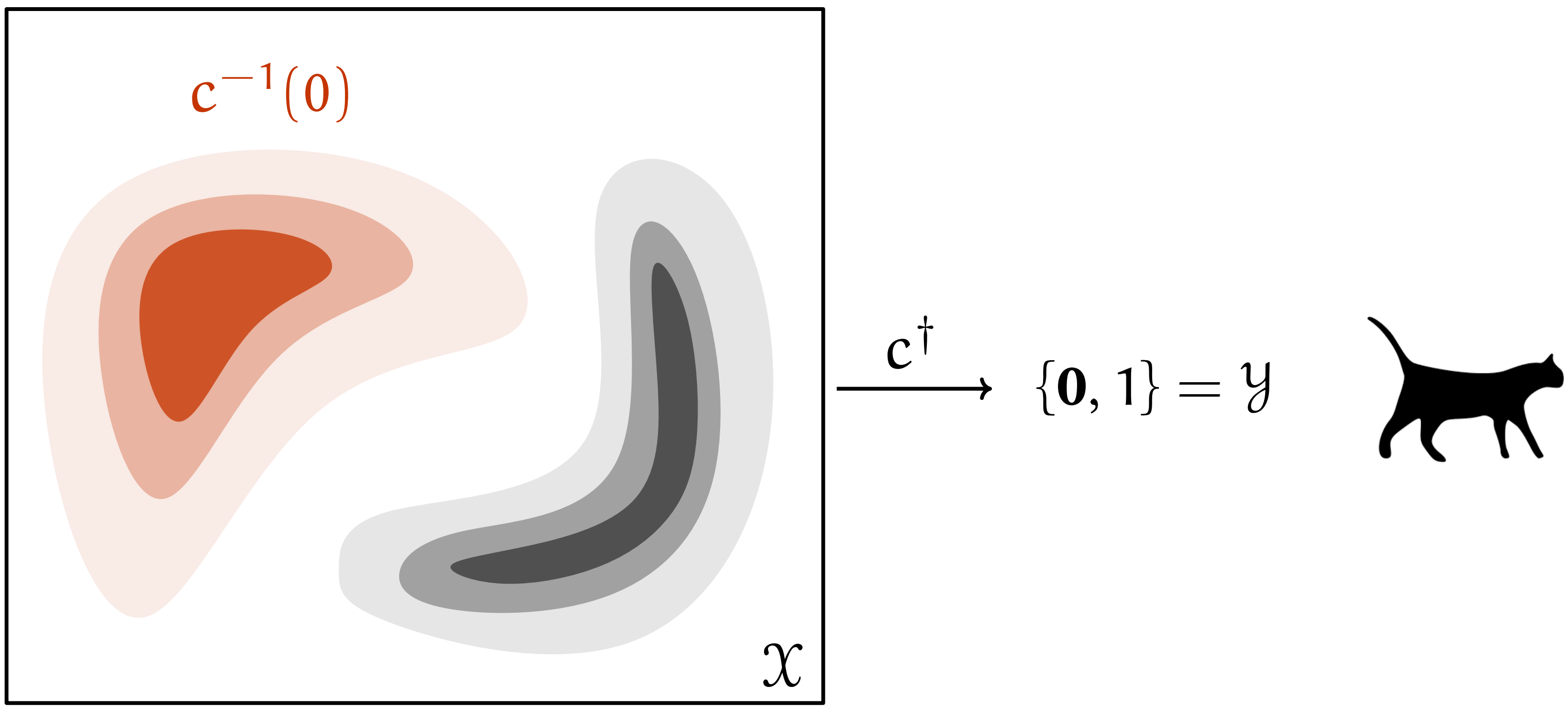
† $c : \mathcal{X} \to \mathcal{Y}$ is an **unknown** labeling function

Application: Topological regularizers

How about **neural classifiers**? [Hofer et al., 2020]

Question: Can we **control** topological properties for generalization

Key idea: encourage "densification" of learned representations



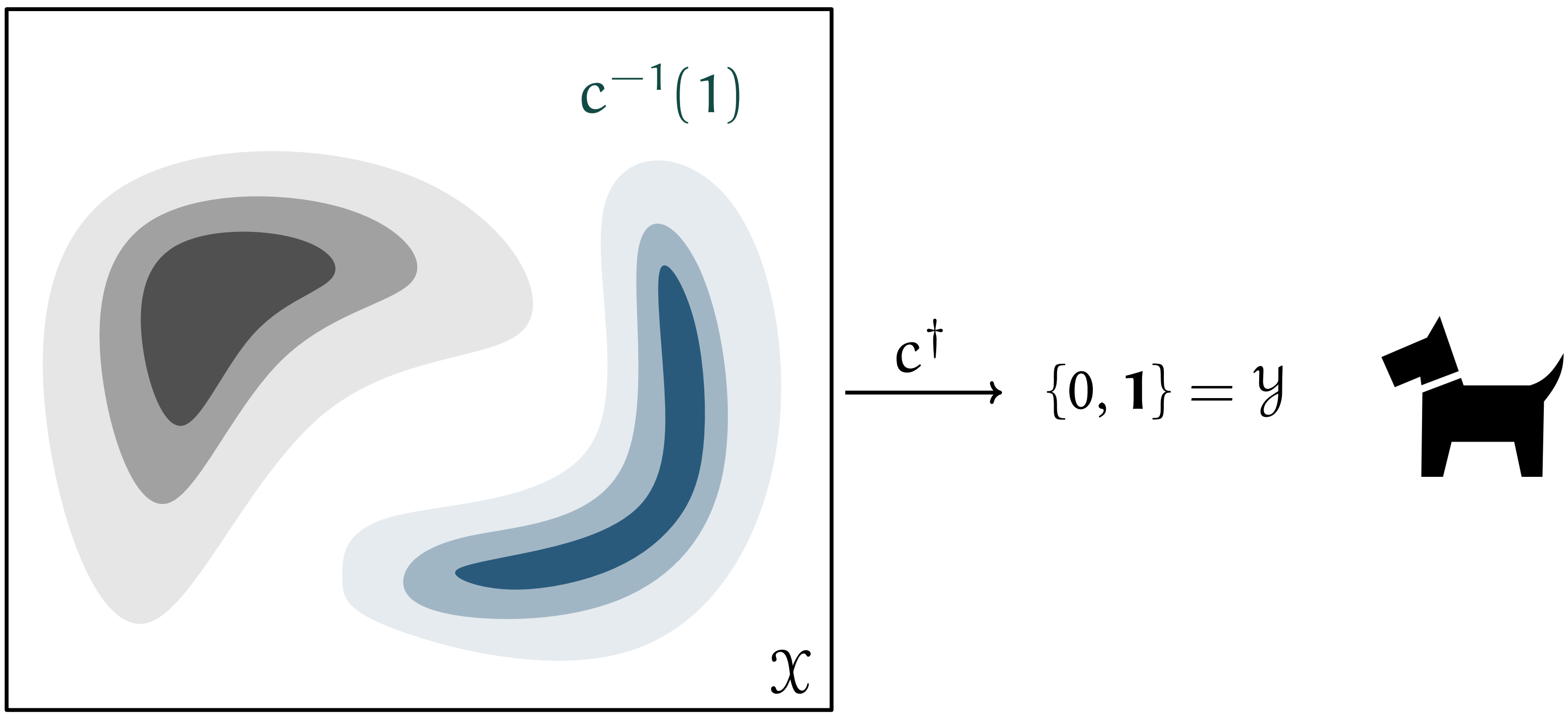
† $c : \mathcal{X} \to \mathcal{Y}$ is an **unknown** labeling function

Application: Topological regularizers

How about **neural classifiers**? [Hofer et al., 2020]

Question: Can we **control** topological properties for generalization

Key idea: encourage “densification” of learned representations



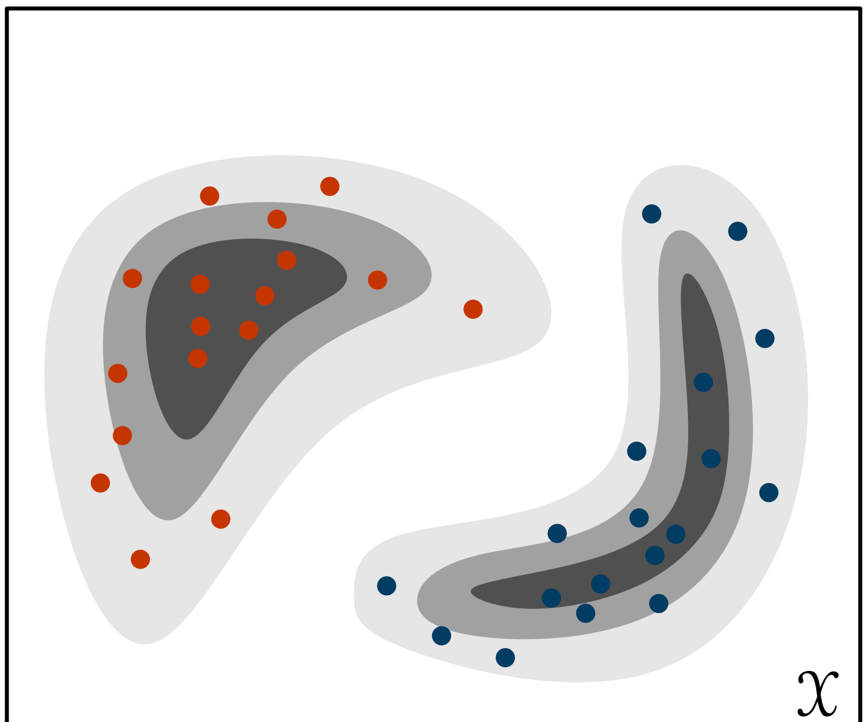
† $c : \mathcal{X} \to \mathcal{Y}$ is an **unknown** labeling function

Application: Topological regularizers

How about **neural classifiers**? [Hofer et al., 2020]

Question: Can we **control** topological properties for generalization

Key idea: encourage "densification" of learned representations



$$\xrightarrow{c^\dagger} \{0, 1\} = \mathcal{Y}$$

We want to **approximate** c by F

(implemented as a neural network)

† $c : \mathcal{X} \to \mathcal{Y}$ is an **unknown** labeling function

Application: Topological regularizers

One aspect of the **generalization puzzle** in deep learning:

Generalization in spite of memorization

Application: Topological regularizers

One aspect of the **generalization puzzle** in deep learning:

Generalization in spite of memorization

In fact, we can typically fit the training data **without error**, i.e., $L_S(F) = 0$.
(even under random labels [Zhang et al., 2017])

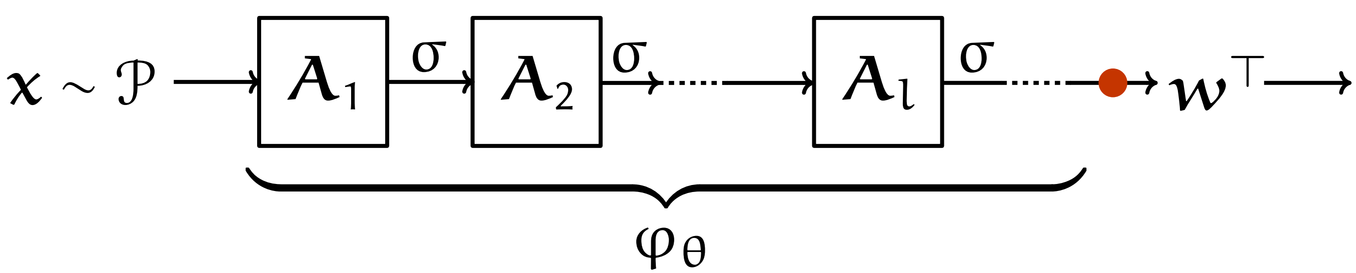
Application: Topological regularizers

One aspect of the **generalization puzzle** in deep learning:

Generalization in spite of memorization

In fact, we can typically fit the training data **without error**, i.e., $L_S(F) = 0$. (even under random labels [Zhang et al., 2017])

Consider



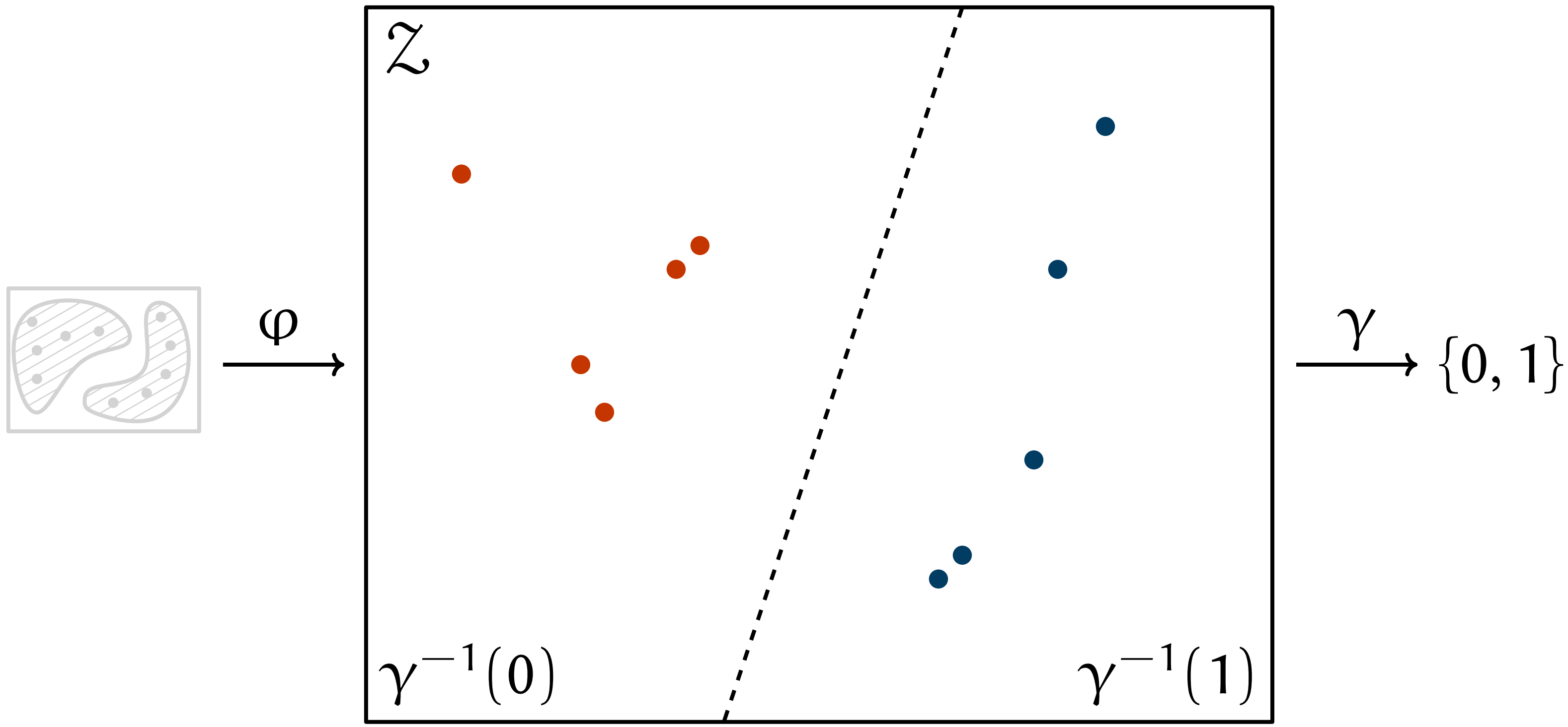
In [Hofer et al., 2020], we study how the distribution around representations of training samples, $\varphi_*(\mathcal{P})$, affects generalization.

Application: Topological regularizers

Lets decompose F as $\mathsf{F} = \gamma \circ \varphi : \mathcal{X} \to \mathcal{Z} \to \mathcal{Y}$ with $\gamma(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{w}^\top \boldsymbol{x})$.

Application: Topological regularizers

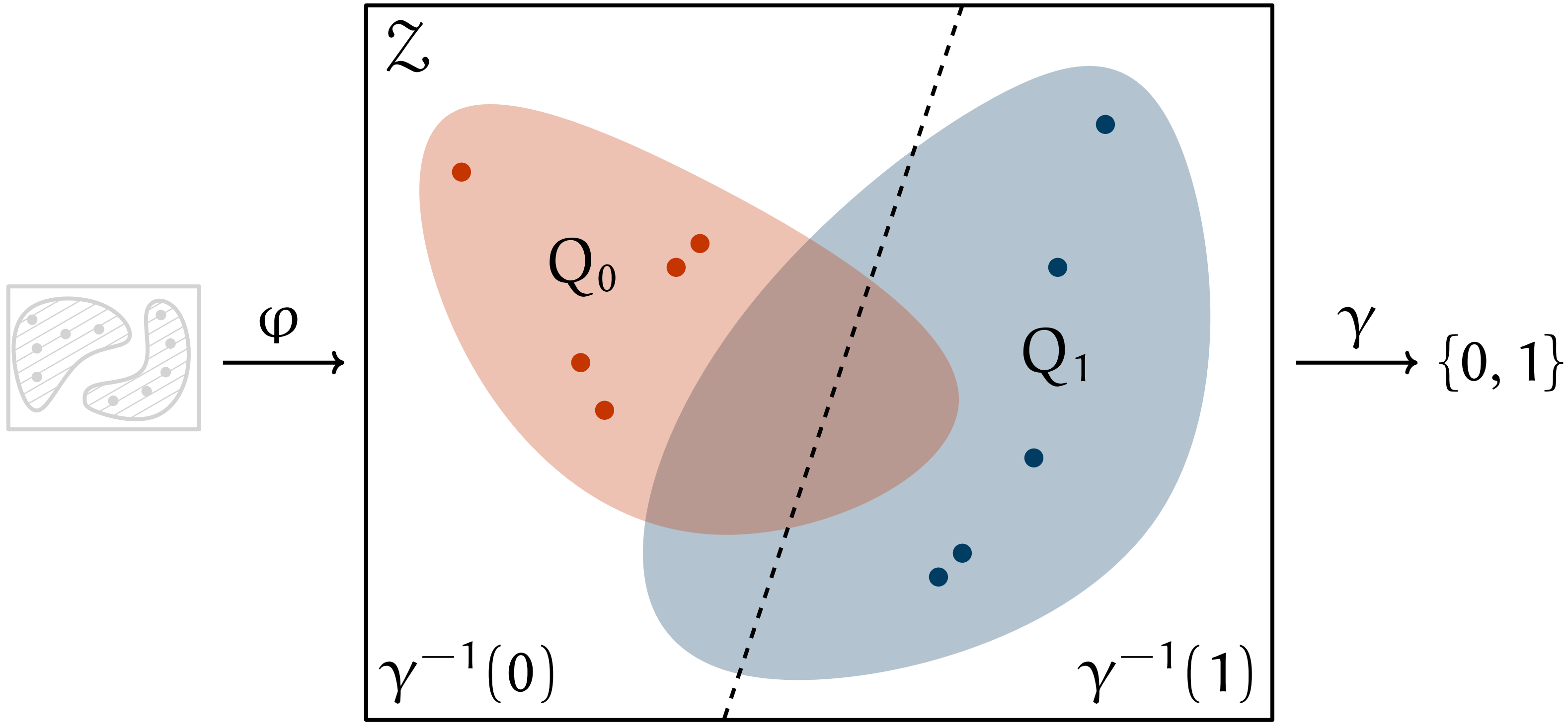
Lets decompose F as $F = \gamma \circ \varphi : \mathcal{X} \to \mathcal{Z} \to \mathcal{Y}$ with $\gamma(\boldsymbol{x}) = \operatorname{sgn}(\boldsymbol{w}^\top \boldsymbol{x})$.



- ▷ $\mathcal{Z}$ is the **codomain** of φ, $\gamma^{-1}(i)$ the **decision region** of class i

Application: Topological regularizers

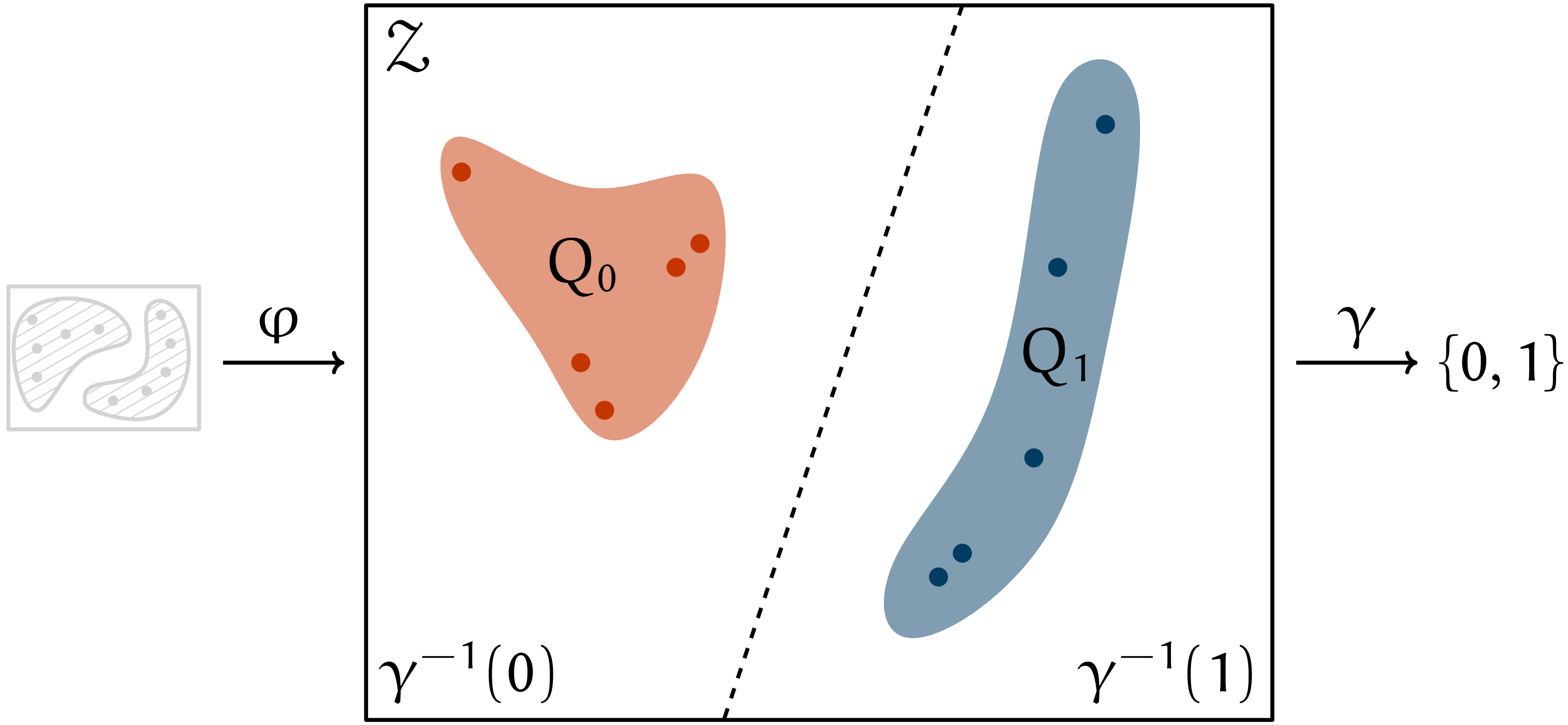
Lets decompose F as $F = \gamma \circ \varphi : \mathcal{X} \to \mathcal{Z} \to \mathcal{Y}$ with $\gamma(\boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{w}^\top \boldsymbol{x})$.



- $\mathcal{Z}$ is the **codomain** of φ, $\gamma^{-1}(i)$ the **decision region** of class i
- **Label-wise** distribution, Q_i (restriction of $\varphi_*(\mathcal{P})$ to class i), in $\mathcal{Z}$

Application: Topological regularizers

Lets decompose F as $F = \gamma \circ \varphi : \mathcal{X} \to \mathcal{Z} \to \mathcal{Y}$ with $\gamma(\boldsymbol{x}) = \text{sgn}(\boldsymbol{w}^\top \boldsymbol{x})$.



- ▷ $\mathcal{Z}$ is the **codomain** of φ, $\gamma^{-1}(i)$ the **decision region** of class i
- ▷ **Label-wise** distribution, Q_i (restriction of $\varphi_*(\mathcal{P})$ to class i), in $\mathcal{Z}$

We aim for a **densification** of Q_i via regularization of φ.

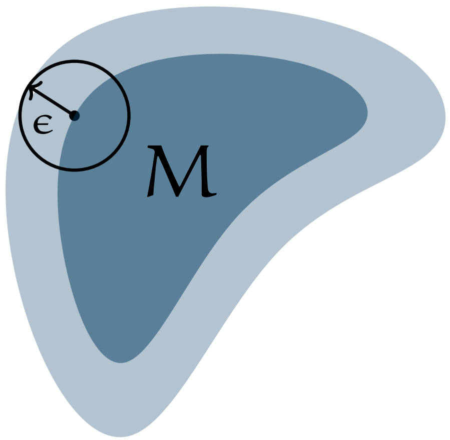
Lets take a closer look at **densification**.

Application: Topological regularizers

Lets take a closer look at **densification**.

Consider, for a reference set $M \subset \mathcal{Z}$, its metric extension†

$$M_\epsilon = \bigcup_{x \in M} B(x, \epsilon), \quad \epsilon > 0$$



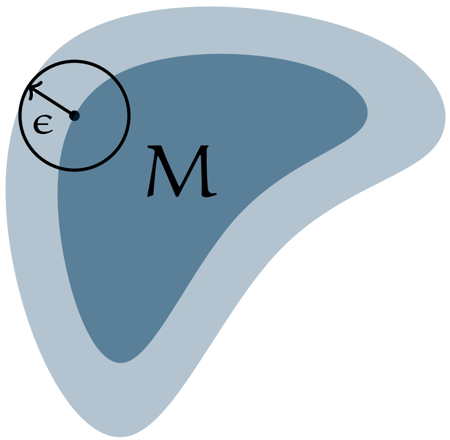
† $B(x, \epsilon) = \{u \in \mathcal{Z} : d(x, u) \leqslant \epsilon\}$

Application: Topological regularizers

Lets take a closer look at **densification**.

Consider, for a reference set $M \subset \mathcal{Z}$, its metric extension[†]

$$M_\epsilon = \bigcup_{x \in M} B(x, \epsilon), \quad \epsilon > 0$$

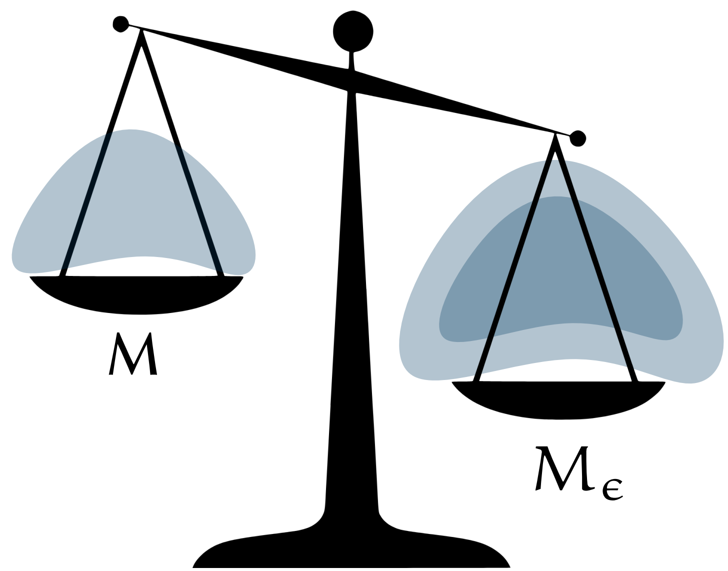


Question: How much mass is in the ϵ-belt?

[†] $B(x, \epsilon) = \{u \in \mathcal{Z} : d(x, u) \leqslant \epsilon\}$

Informally, **densification** means:

For a given mass in the reference set M, increase the mass concentrated in its ϵ-extension!



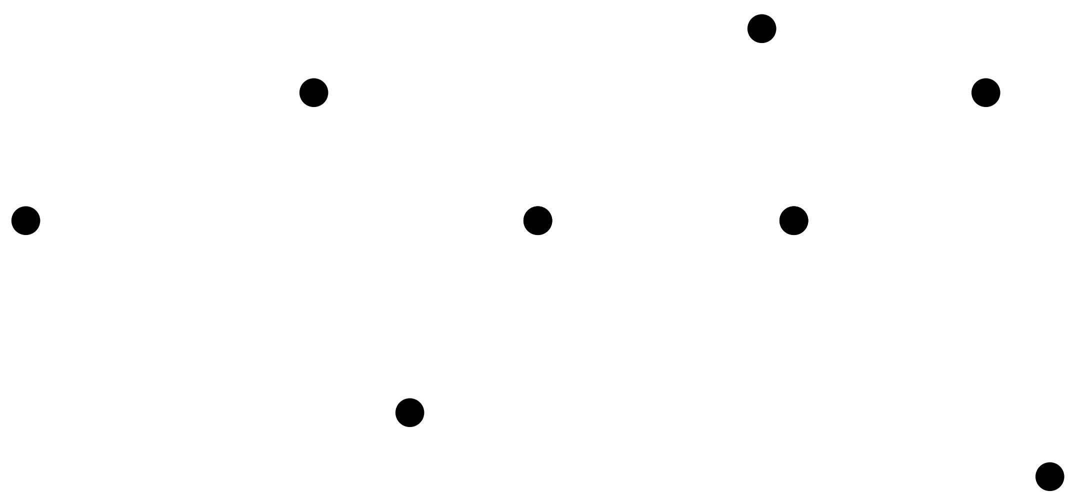
Application: Topological regularizers

The **idea** is to exert control over connectivity properties!

Application: Topological regularizers

The **idea** is to exert control over connectivity properties!

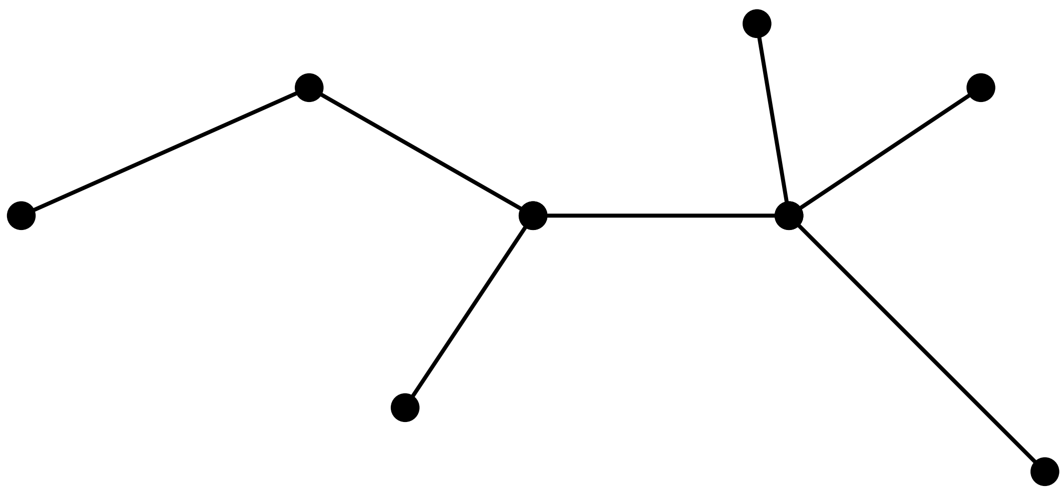
Consider the (Euclidean) **minimal spanning tree (*MST*)**[†]:



Application: Topological regularizers

The **idea** is to exert control over connectivity properties!

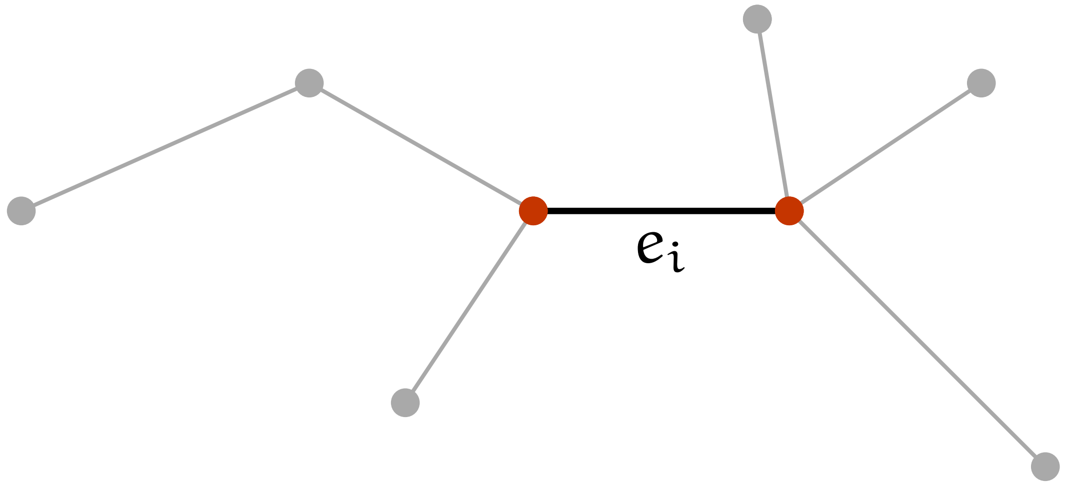
Consider the (Euclidean) **minimal spanning tree (MST)**[†]:



Application: Topological regularizers

The **idea** is to exert control over connectivity properties!

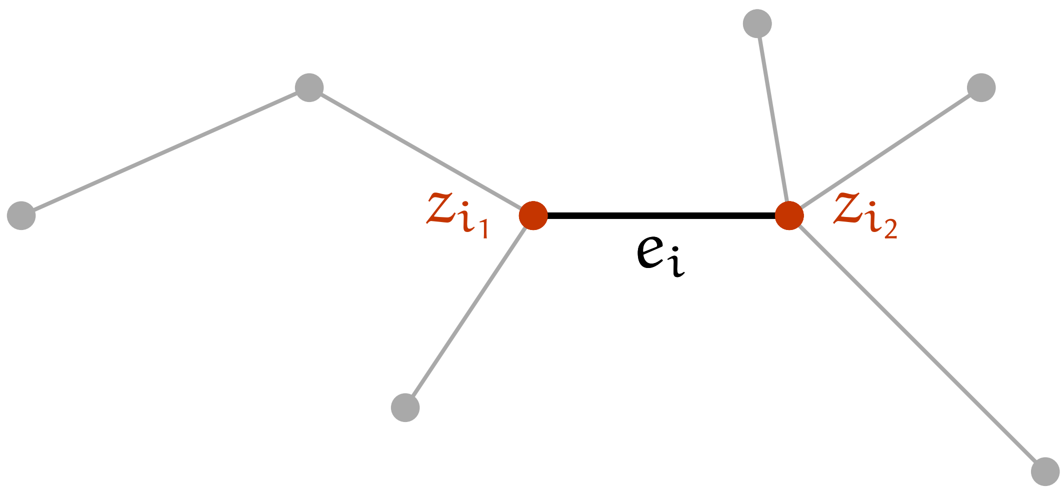
Consider the (Euclidean) **minimal spanning tree (MST)**[†]:



Application: Topological regularizers

The **idea** is to exert control over connectivity properties!

Consider the (Euclidean) **minimal spanning tree (MST)**†:



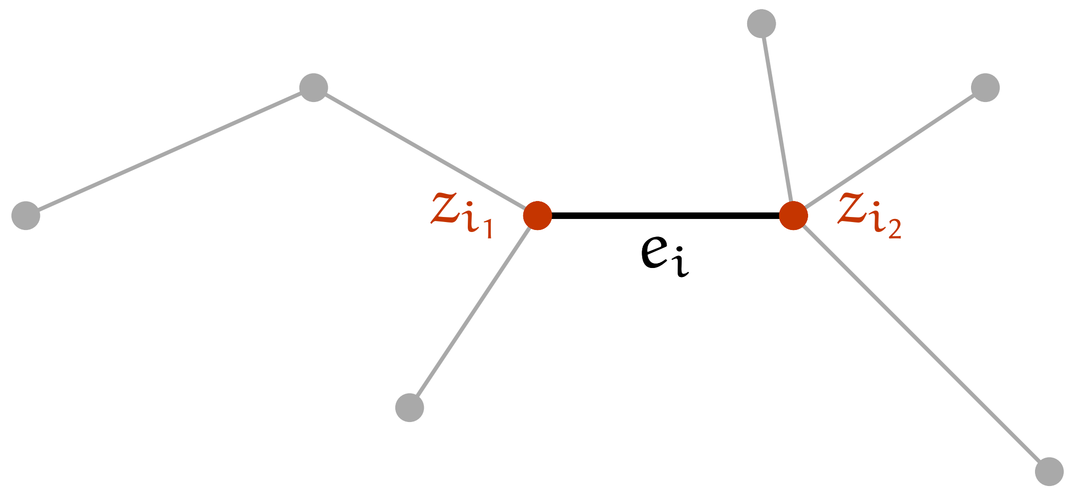
$$\mathrm{len}(e_i) = d(z_{i_1}, z_{i_2})$$

† $d(x, y) = \|x - y\|$

Application: Topological regularizers

The **idea** is to exert control over connectivity properties!

Consider the (Euclidean) **minimal spanning tree (MST)**[†]:



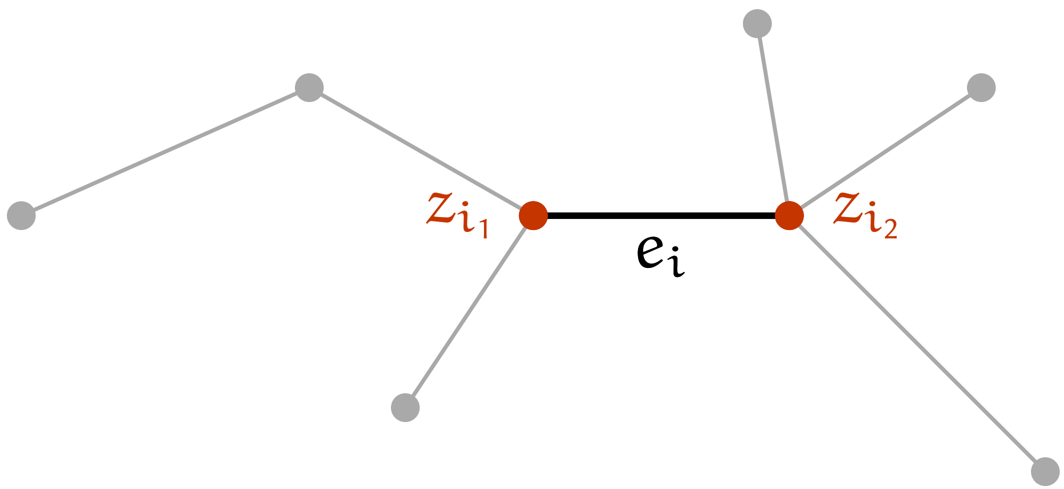
$$\mathrm{len}(e_i) = d(\varphi(x_{i_1}), \varphi(x_{i_2}))$$

[†] $d(x, y) = \|x - y\|$

Application: Topological regularizers

The **idea** is to exert control over connectivity properties!

Consider the (Euclidean) **minimal spanning tree (MST)**†:



as φ is parametrized by a **neural network** with parameters θ

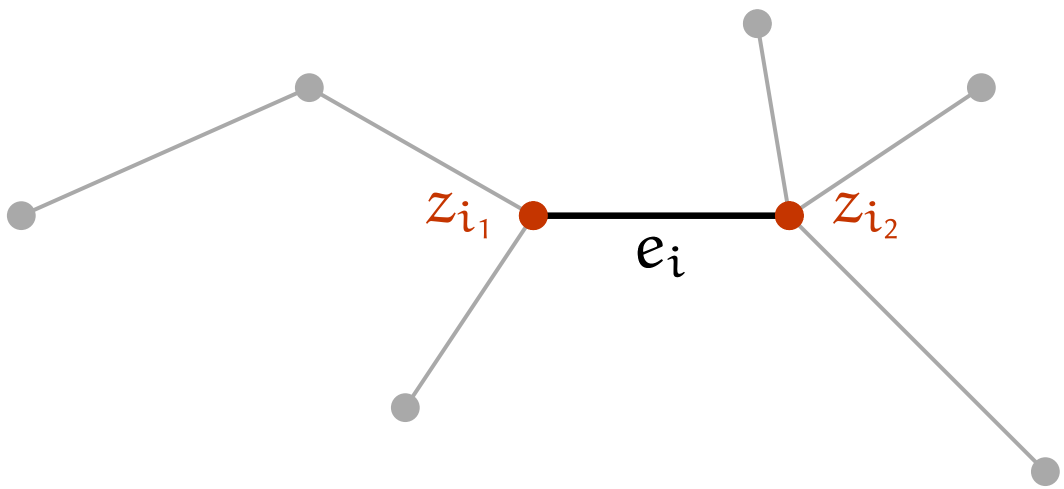
$$\text{len}(e_i) = d(\varphi_\theta(x_{i_1}), \varphi_\theta(x_{i_2}))$$

† $d(x, y) = \|x - y\|$

Application: Topological regularizers

The **idea** is to exert control over connectivity properties!

Consider the (Euclidean) **minimal spanning tree (MST)**[†]:



as φ is parametrized by a **neural network** with parameters θ

$$\mathrm{len}(e_i) = d(\varphi_\theta(x_{i_1}), \varphi_\theta(x_{i_2}))$$

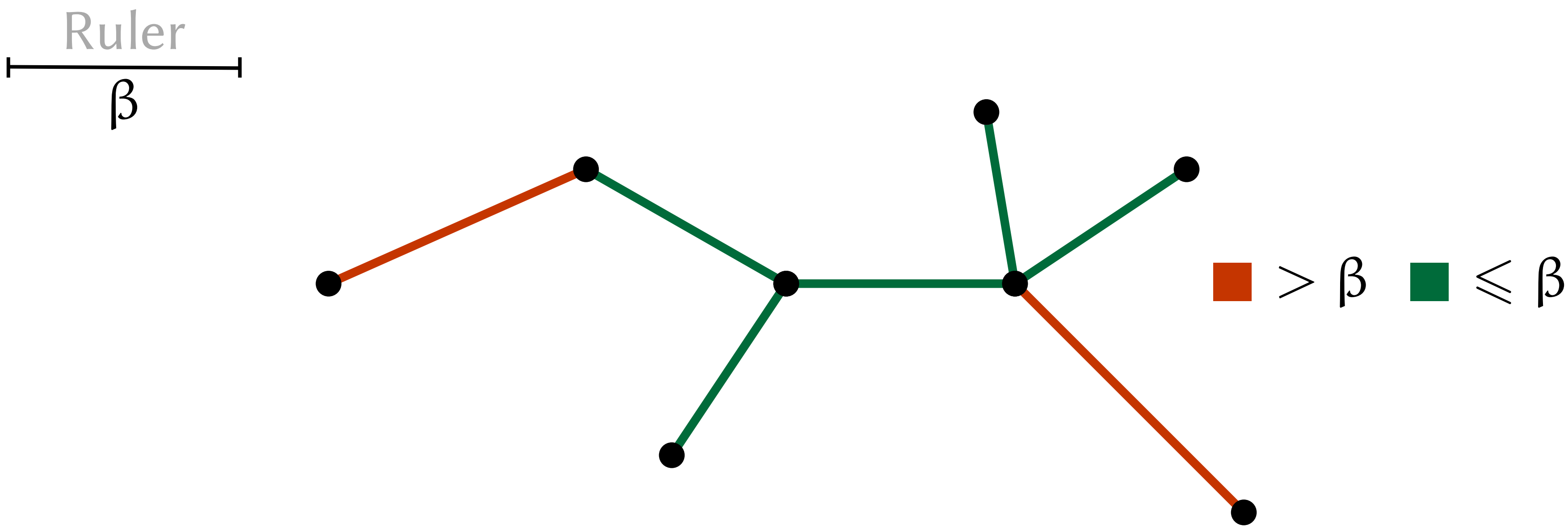
Differentiable in θ

$\Rightarrow$ we can control the **edge lengths** of the MST (as mentioned earlier)

[†] $d(x, y) = \|x - y\|$

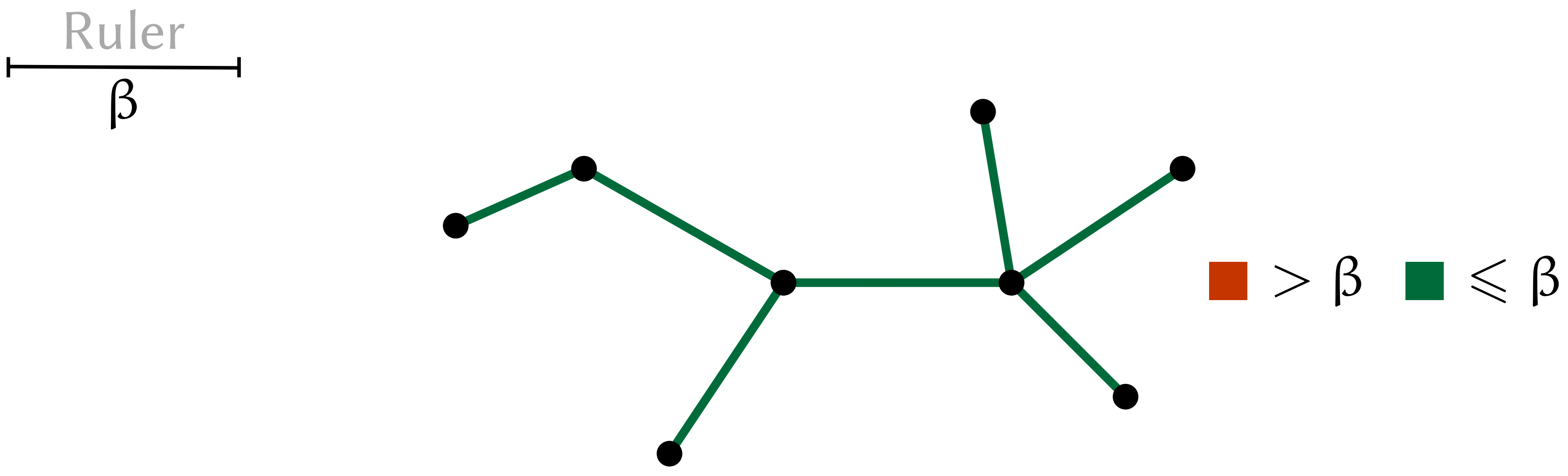
Application: Topological regularizers

We call $z_1, \ldots, z_b \in \mathcal{Z}$ β-connected if all edges in the corresponding MST are not longer than β.



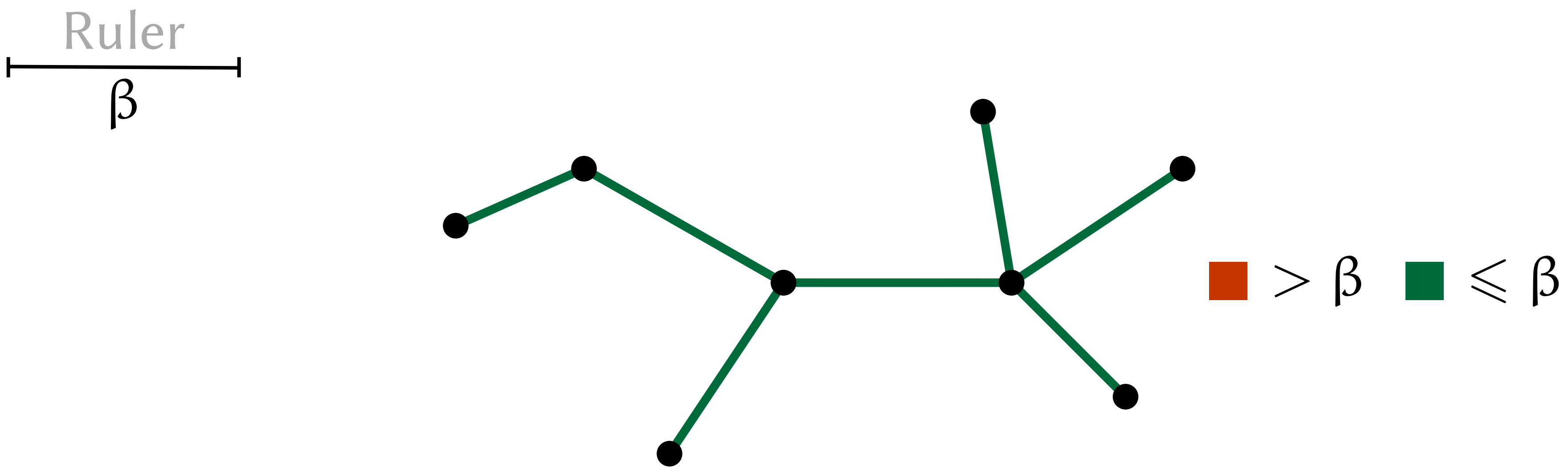
Application: Topological regularizers

We call $z_1, \ldots, z_b \in \mathcal{Z}$ β-connected if all edges in the corresponding MST are not longer than β.



Application: Topological regularizers

We call $z_1, \ldots, z_b \in \mathcal{Z}$ β-connected if all edges in the corresponding MST are not longer than β.



This allows us to talk about properties of $z_1, \ldots, z_b \sim Q$, i.e., b iid draws from Q.

Application: Topological regularizers

Let $b \in \mathbb{N}$. We call Q a c_b^β-connected distribution if

$$c_b^\beta \leqslant \mathbf{Pr}[Z_1, \ldots, Z_b \text{ are } \beta\text{-connected}]$$

holds for $Z_1, \ldots, Z_b \overset{\text{iid}}{\sim} Q$ with $\beta > 0$, $c_b^\beta > 0$.

Let $b \in \mathbb{N}$. We call Q a c_b^β-connected distribution if

$$c_b^\beta \leqslant \mathbf{Pr}[Z_1, \ldots, Z_b \text{are } \beta\text{-connected}]$$

holds for $Z_1, \ldots, Z_b \overset{\text{iid}}{\sim} Q$ with $\beta > 0$, $c_b^\beta > 0$.

This is a property of the **product measure** Q^b.

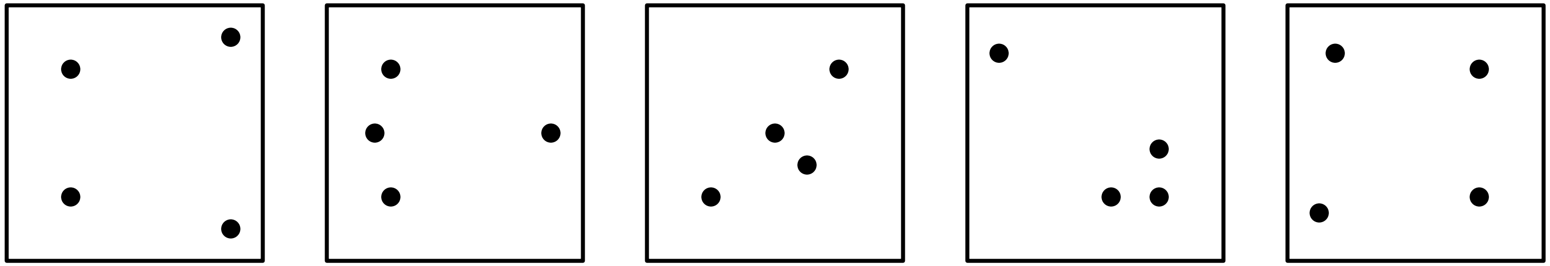
Let $b \in \mathbb{N}$. We call Q a c_b^β-connected distribution if

$$c_b^\beta \leqslant \mathbf{Pr}[Z_1, \ldots, Z_b \text{ are } \beta\text{-connected}]$$

holds for $Z_1, \ldots, Z_b \overset{\text{iid}}{\sim} Q$ with $\beta > 0$, $c_b^\beta > 0$.

This is a property of the **product measure** Q^b.

Example: five draws from Q^b with $b = 4$



Application: Topological regularizers

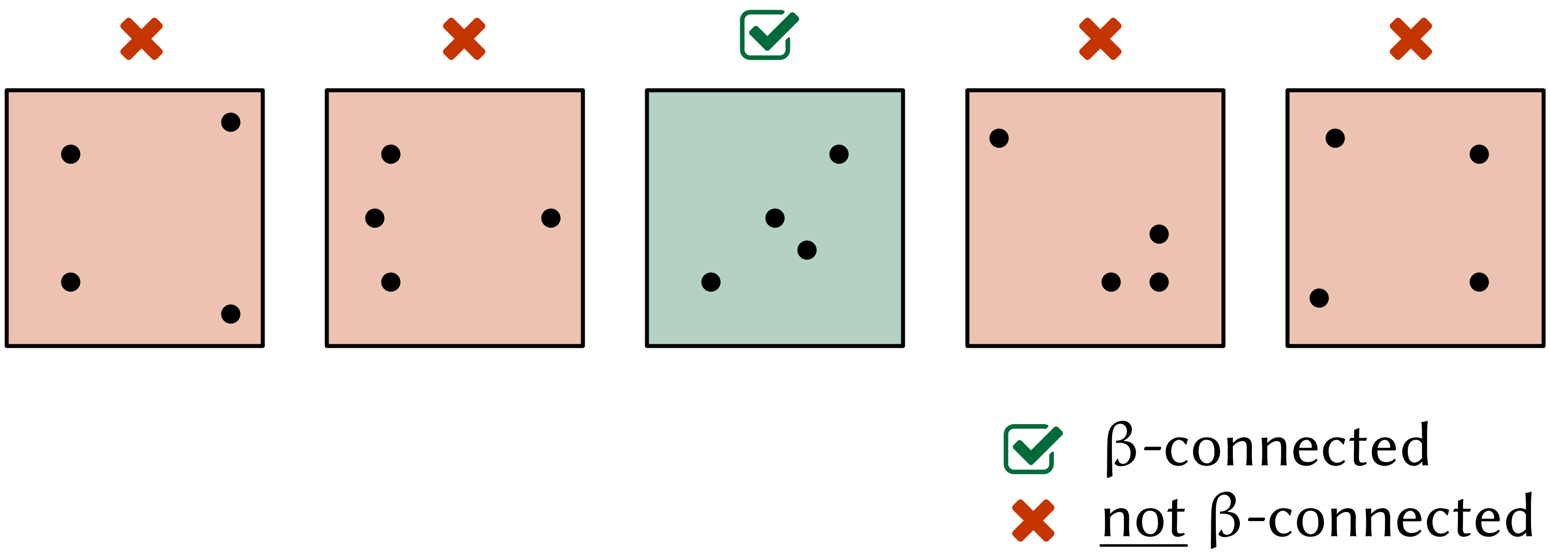
Let $b \in \mathbb{N}$. We call Q a c_b^β-connected distribution if

$$c_b^\beta \leqslant \mathbf{Pr}[Z_1, \ldots, Z_b \text{ are } \beta\text{-connected}]$$

holds for $Z_1, \ldots, Z_b \overset{\text{iid}}{\sim} Q$ with $\beta > 0$, $c_b^\beta > 0$.

This is a property of the **product measure** Q^b.

Example: five draws from Q^b with $b = 4$



Application: Topological regularizers

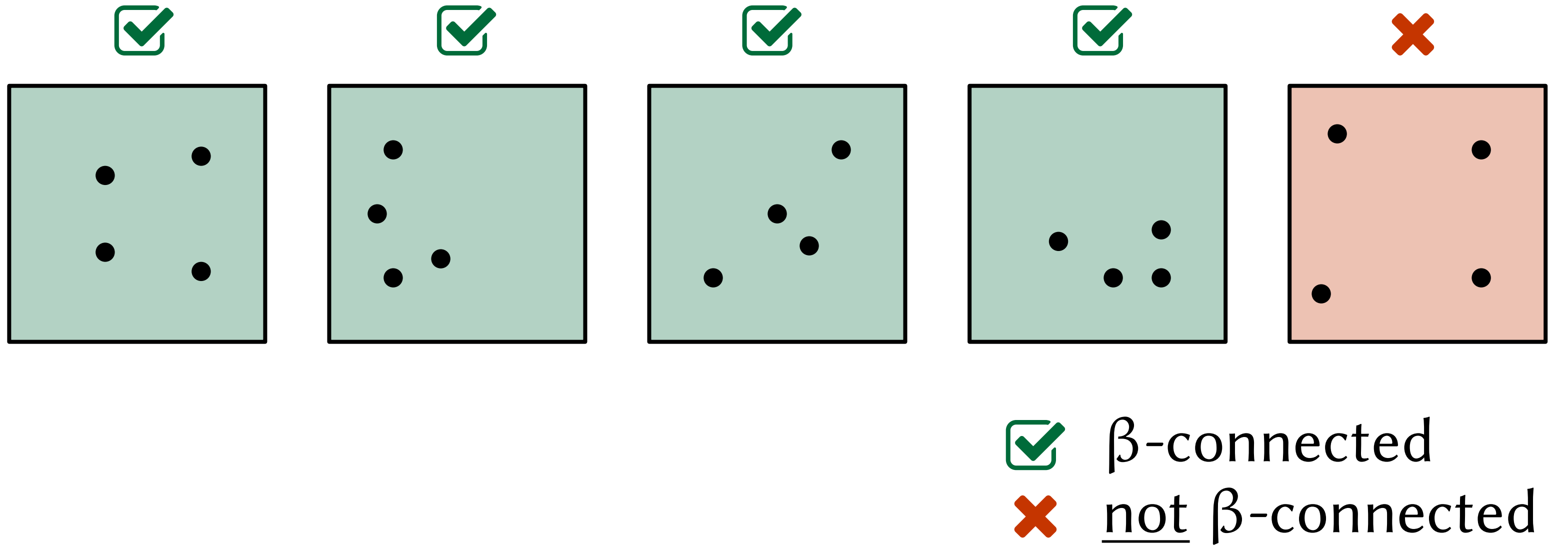
Let $b \in \mathbb{N}$. We call Q a c_b^β-connected distribution if

$$c_b^\beta \leqslant \mathbf{Pr}[Z_1, \ldots, Z_b \text{ are } \beta\text{-connected}]$$

holds for $Z_1, \ldots, Z_b \overset{\text{iid}}{\sim} Q$ with $\beta > 0$, $c_b^\beta > 0$.

This is a property of the **product measure** Q^b.

Example: five draws from Q^b with $b = 4$



Application: Topological regularizers

Let $b \in \mathbb{N}$. We call Q a c_b^β-connected distribution if

$$c_b^\beta \leqslant \mathbf{Pr}[Z_1, \ldots, Z_b \text{ are } \beta\text{-connected}]$$

holds for $Z_1, \ldots, Z_b \overset{\text{iid}}{\sim} Q$ with $\beta > 0$, $c_b^\beta > 0$.

1. We can show that controlling connectivity properties (β-connectedness) of Q^b leads to densification of Q.

2. We can show that densification directly relates to generalization.

✓ β-connected
✗ not β-connected

Application: Topological regularizers

Some results for a neural classifier[‡] on **MNIST** (10 classes) in a **small sample-size** regime (250 samples):

Vanilla	7.1 +/- 1.0

[‡] using a mid-size convolutional neural network (CNN13)

Application: Topological regularizers

Some results for a neural classifier[‡] on **MNIST** (10 classes) in a **small sample-size** regime (250 samples):

Vanilla	7.1 +/- 1.0
+ Jacobian reg.	6.2 +/- 0.8
+ DeCov	6.5 +/- 1.1
+ VR	6.1 +/- 0.5
+ cw-CR	7.0 +/- 0.6
+ cw-VR	6.2 +/- 0.8

[‡] using a mid-size convolutional neural network (CNN13)

Application: Topological regularizers

Some results for a neural classifier‡ on **MNIST** (10 classes) in a **small sample-size** regime (250 samples):

Vanilla	7.1 +/- 1.0
+ Jacobian reg.	6.2 +/- 0.8
+ DeCov	6.5 +/- 1.1
+ VR	6.1 +/- 0.5
+ cw-CR	7.0 +/- 0.6
+ cw-VR	6.2 +/- 0.8
+ ConnLoss (best)	5.6 +/- 0.7
+ ConnLoss†	5.9 +/- 0.3

† β chosen via cross-validation on a small validation set

‡ using a mid-size convolutional neural network (CNN13)

Application: Topological regularizers

Some results for a neural classifier[‡] on **CIFAR10** (10 classes) in a **small sample-size** regime (500 samples):

`Vanilla`	`39.4 +/- 1.5`

[‡] using a mid-size convolutional neural network (CNN13)

Application: Topological regularizers

Some results for a neural classifier[‡] on **CIFAR10** (10 classes) in a **small sample-size** regime (500 samples):

Vanilla	39.4 +/- 1.5
+ Jacobian reg.	39.7 +/- 2.0
+ DeCov	38.2 +/- 1.5
+ VR	38.6 +/- 1.4
+ cw-CR	39.0 +/- 1.9
+ cw-VR	38.5 +/- 1.6

‡ using a mid-size convolutional neural network (CNN13)

Application: Topological regularizers

Some results for a neural classifier[‡] on **CIFAR10** (10 classes) in a **small sample-size** regime (500 samples):

Vanilla	39.4 +/- 1.5
+ Jacobian reg.	39.7 +/- 2.0
+ DeCov	38.2 +/- 1.5
+ VR	38.6 +/- 1.4
+ cw-CR	39.0 +/- 1.9
+ cw-VR	38.5 +/- 1.6
+ ConnLoss (best)	36.5 +/- 1.2
+ ConnLoss[†]	36.8 +/- 0.3

[†] β chosen via cross-validation on a small validation set

[‡] using a mid-size convolutional neural network (CNN13)

What's ahead of us?

There is so much exciting stuff that is going on right now!

Here are **some examples** . . .

- ▷ Theory for for optimizing PH-based functions [Carrière et al., 2020]
- ▷ Studying learning behavior of neural networks [Rieck et al., 2018]
- ▷ PH for learning with graphs [Hofer et al., 2019; Rieck et al. 2021]
- ▷ Using simplicial complexes for message passing [Bodnar et al., 2021]
- ▷ Differentiable topology layers [Brüel-Gabrielsson et al., 2019]
- ▷ Topological attention for time-series forecasting [Zeng et al., 2021]
- ▷ Topology-preserving image segmentation [Hu et al., 2019]
- ▷ Topological regularization of decision boundaries [Chen et al., 2019]

Again, this is, by far, **not** an exhaustive listing!

What I (personally) find interesting

Continuing work along the lines of [Bianchini & Scarselli, 2014], i.e., using concepts from topology to study **hypothesis set complexity**.

see also [Ramamurthy et al., 2019]
[Guss & Salakhutdinov, 2018]

What I (personally) find interesting

Continuing work along the lines of [Bianchini & Scarselli, 2014], i.e., using concepts from topology to study **hypothesis set complexity**.

see also [Ramamurthy et al., 2019]
[Guss & Salakhutdinov, 2018]

Can we possibly come up with other/better measures of quantifying hypothesis set complexity (similar to VC-dim., or Rademacher complexity)?

What I (personally) find interesting

Continuing work along the lines of [Bianchini & Scarselli, 2014], i.e., using concepts from topology to study **hypothesis set complexity**.

see also [Ramamurthy et al., 2019]
[Guss & Salakhutdinov, 2018]

Can we possibly come up with other/better measures of quantifying hypothesis set complexity (similar to VC-dim., or Rademacher complexity)?

With differentiable layers for NN's that compute PH, we have a great tool – but, we do not really know what to do with it (yet).

Collaborators

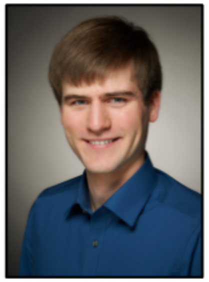
Marc Niethammer
UNC Chapel Hill
@MarcNiethammer

Jan Reininghaus
IST Austria (back then)

Bastian Rieck
ETH
@Pseudomanifold

Ulrich Bauer
TUM

Florian Graf
Univ. Salzburg



Chris Hofer
Univ. Salzburg

Stefan Huber
IST Austria (back then)
@shuber3

Thank You!