Topological Data Analysis for Deep Learning

Rubén Ballester Bautista
Topological Machine Learning @ UB

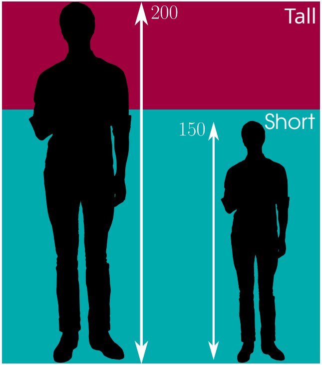
4 June 2024

Table of Contents

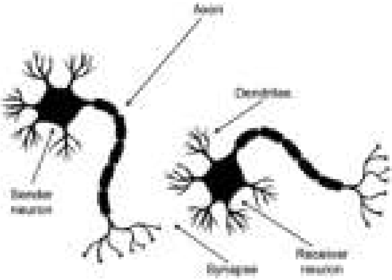
Deep Learning

Introduction to supervised ML

- **Goal**: Approximate an unknown function $f : X \to Y$ from a sample of points $\mathcal{D} = \{(x_i, f(x_i))\}_{i=1}^m$ with $x_i \sim \mathbb{P}_X$. Usually, with $X = \mathbb{R}^{d_i}$.
- In this talk, depending on the codomain Y we can distinguish between **classification** ($Y = \{1, 2, \dots, L\}$) and **regression** ($Y = \mathbb{R}^{d_e}$).



Deep learning (I)



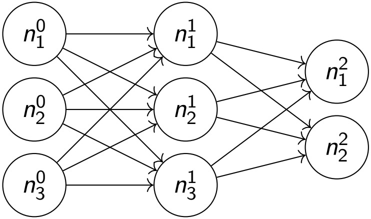
- Neural networks were originally inspired by the brain's structure.
- At one end, a sender neuron sends a signal to the next neuron, which travels through the axon and reaches the dendrites of the receiver using the synapses at the end of the axon.
- This communication can be represented by a **graph**.

$$S \longrightarrow R$$

Image from Sivadas A and Broadie K (2020) *How Does My Brain Communicate With My Body?*. Front. Young Minds. 8:540970. doi: 10.3389/frym.2020.540970

Deep learning (II)

- In the brain, there are many neurons that are interconnected in a complex manner.



- These kind of graphs define the most basic neural networks, called **feedforward neural networks**.

A first definition

Definition 1: Let $L \in \mathbb{N}$. A **feedforward neural network** is a function $\phi\colon \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ defined recursively as a composition of L functions $\phi^{(l)}\colon \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$, $l \in \{1, \dots, L\}$, as follows:

$$\bar{\phi}^{(l)}(x) = \begin{cases} W^{(1)}x + b^{(1)} & \text{if } l = 1, \\ W^{(l)}\phi^{(l-1)}(x) + b^{(l)} & \text{if } l \in \{2, \dots, L\}, \end{cases}$$

$$\phi^{(l)}(x) = \varphi^{(l)}\left(\bar{\phi}^{(l)}(x)\right) \text{ for } l \in \{1, \dots, L\},$$

$$\phi(x) = \phi^{(L)}(x),$$

where $W^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$ and $b^{(l)} \in \mathbb{R}^{N_l}$ are the weights and biases of the network, respectively, and $\varphi^{(l)}$ is a non-linear activation function.

A first definition

Definition 1: Let $L \in \mathbb{N}$. A **feedforward neural network** is a function $\phi\colon \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ defined recursively as a composition of L functions $\phi^{(l)}\colon \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$, $l \in \{1, \ldots, L\}$, as follows:

$$\bar{\phi}^{(l)}(x) = \begin{cases} W^{(1)}x + b^{(1)} & \text{if } l = 1, \\ W^{(l)}\phi^{(l-1)}(x) + b^{(l)} & \text{if } l \in \{2, \ldots, L\}, \end{cases}$$

$$\phi^{(l)}(x) = \varphi^{(l)}\left(\bar{\phi}^{(l)}(x)\right) \text{ for } l \in \{1, \ldots, L\},$$

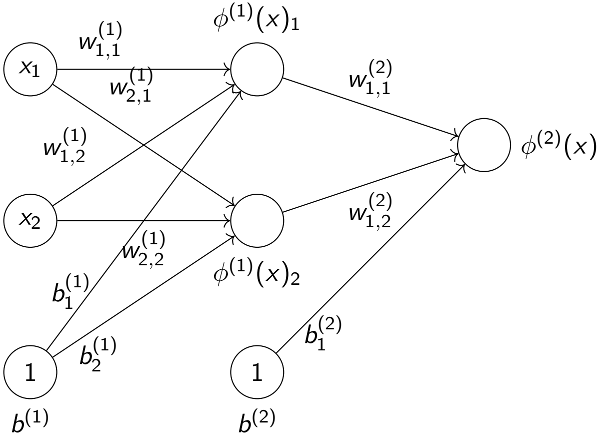
$$\phi(x) = \phi^{(L)}(x),$$

where $W^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$ and $b^{(l)} \in \mathbb{R}^{N_l}$ are the weights and biases of the network, respectively, and $\varphi^{(l)}$ is a non-linear activation function.

- For a classification problem with $Y = \{1, \ldots, N_L\}$, we can use the **argmax** function at the end of the network to obtain a label in Y.

An example

- Take $N_0 = 2$, $N_1 = 2$, $N_2 = 1$ and $\varphi^{(l)}(x) = \max\{0, x\}$. Take $W^{(l)} = \left(w^{(l)}_{i,j}\right)_{i,j} = (i+j)_{i,j}$ and $b^{(l)} = l$ for all l.



Training neural networks

- How do we determine the weights and biases of a neural network so that it approximates the function $f : X \to Y$ as desired?

Training neural networks

- How do we determine the weights and biases of a neural network so that it approximates the function $f : X \to Y$ as desired?
- First, the weights and biases are initialized **randomly**.

Training neural networks

- How do we determine the weights and biases of a neural network so that it approximates the function $f : X \to Y$ as desired?
- First, the weights and biases are initialized **randomly**.
- Then, the weights and biases of a neural network are iteratively refined using **gradient descent** (or similar) to minimize an objective function $\mathcal{L}$ called **loss**.

Training neural networks

UNIVERSITAT DE BARCELONA

- How do we determine the weights and biases of a neural network so that it approximates the function $f : X \to Y$ as desired?
- First, the weights and biases are initialized **randomly**.
- Then, the weights and biases of a neural network are iteratively refined using **gradient descent** (or similar) to minimize an objective function $\mathcal{L}$ called **loss**.
- For a classification problem, a common loss function is the **cross-entropy loss**:

$$\mathcal{L}(\phi, \mathcal{D}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{N_L} \log \frac{\exp(\phi(x_i)_j)}{\sum_{k=1}^{C} \exp(\phi(x_i)_k)} \mathbb{1}(y_i = N_j).$$

Training neural networks

- How do we determine the weights and biases of a neural network so that it approximates the function $f : X \to Y$ as desired?
- First, the weights and biases are initialized **randomly**.
- Then, the weights and biases of a neural network are iteratively refined using **gradient descent** (or similar) to minimize an objective function $\mathcal{L}$ called **loss**.
- For a classification problem, a common loss function is the **cross-entropy loss**:

$$\mathcal{L}(\phi, \mathcal{D}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{N_L} \log \frac{\exp(\phi(x_i)_j)}{\sum_{k=1}^{C} \exp(\phi(x_i)_k)} \mathbb{1}(y_i = N_j).$$

- For a regression problem, a common loss function is the **mean squared error**:

$$\mathcal{L}(\phi, \mathcal{D}) = \frac{1}{m} \sum_{i=1}^{m} (\phi(x_i) - y_i)^2 \,.$$

Gradient descent (I)

- The **gradient descent** algorithm is an iterative optimization algorithm that uses the gradient of the loss function to update the weights and biases of a neural network.

Gradient descent (I)

- The **gradient descent** algorithm is an iterative optimization algorithm that uses the gradient of the loss function to update the weights and biases of a neural network.
- The weights and biases are updated as follows:

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial W^{(l)}},$$
$$b^{(l)} \leftarrow b^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial b^{(l)}},$$

where α is the **learning rate**.

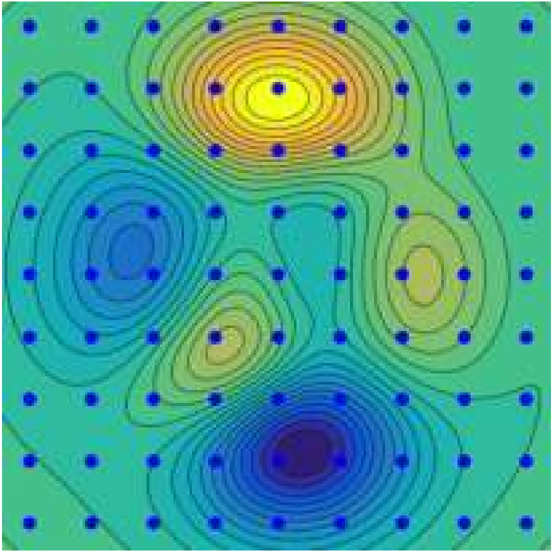
Gradient descent (I)

- The **gradient descent** algorithm is an iterative optimization algorithm that uses the gradient of the loss function to update the weights and biases of a neural network.
- The weights and biases are updated as follows:

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial W^{(l)}},$$
$$b^{(l)} \leftarrow b^{(l)} - \alpha \frac{\partial \mathcal{L}}{\partial b^{(l)}},$$

  where α is the **learning rate**.
- The gradient of the loss function is computed using the **backpropagation** algorithm.

Gradient descent (II)



Effectivity measures and generalization

UNIVERSITAT DE BARCELONA

- Once a neural network has been trained, we evaluate its performance. We use a different dataset called the **test set**, denoted by $\mathcal{D}_{\text{test}}$.
- **Effectivity measures** are used to evaluate the performance of a neural network. For classification problems, we can use **accuracy**:

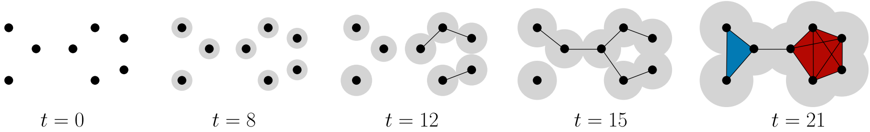
$$\text{Accuracy}(\mathcal{D}, \phi) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \mathbb{1}(\text{argmax}(\phi(x_i)) = y_i).$$

Generalization gap = Train acc. - test acc. = 10%

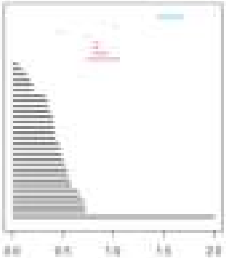
Train accuracy: 60% Test accuracy: 50%

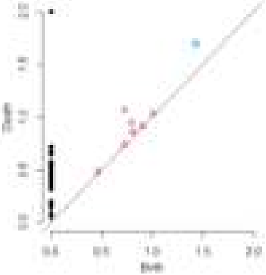
Topological Data Analysis

Persistent homology



$$
\begin{tikzcd}
C_1 \arrow[r] \arrow[d, "H_k"] & C_2 \arrow[r] \arrow[d, "H_k"] & C_3 \arrow[r] \arrow[d, "H_k"] & \cdots \arrow[r] & C_n \arrow[d, "H_k"] \\
H_k(C_1) \arrow[r] & H_k(C_2) \arrow[r] & H_k(C_3) \arrow[r] & \cdots \arrow[r] & H_k(C_n)
\end{tikzcd}
$$





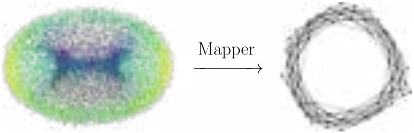
Mapper

Extraction of **simplicial complexes** from point clouds C.

Require: $\mathcal{D}$ with $|D| = m$, filter function $f\colon \mathcal{D} \to \mathbb{R}^d$, finite cover $\mathcal{U} = \{\mathcal{U}_i\}_{i\in I}$ of $\mathrm{Im}(f) \subseteq \mathbb{R}^d$, clustering algorithm $\mathcal{C}$ (e.g. `DBSCAN`).

Ensure: Simplicial complex $S_{\mathcal{D}}$.

$S_{\mathcal{D}} \leftarrow \emptyset$; $\mathcal{D}_i \leftarrow f^{-1}(\mathcal{U}_i)$ for all $i \in I$
for all $i \in I$ **do**
  $\{C_i^1, \dots, C_i^{k_i}\} \leftarrow \mathcal{C}(\mathcal{D}_i)$ {Apply the clustering algorithm to $\mathcal{D}_i$}
  $S_{\mathcal{D}} \leftarrow S_{\mathcal{D}} \cup \{C_i^1, \dots, C_i^{k_i}\}$ {Add the clusters found as vertices}
end for
for all $\{C_1, \dots, C_t\} \in \mathcal{P}\left(\bigcup_{i\in I}\{C_i^1, \dots, C_i^{k_i}\}\right)$ {$\forall$ subsets of found clusters} **do**
  if $\bigcap_{j=1}^{t} C_j \neq \emptyset$ **then**
    $S_{\mathcal{D}} \leftarrow S_{\mathcal{D}} \cup \{\{C_1, \dots, C_t\}\}$ {We add the simplex $\{C_1, \dots, C_t\}$}
  end if
end for
return $S_{\mathcal{D}}$



TDA for Deep Learning

Generalization and persistent homology (I)

- **Objective**: Link the generalization gap of a neural network with the topological properties of the network and the training data. We follow Ballester et al. (2024).[1]

[1] Rubén Ballester et al. "Predicting the generalization gap in neural networks using topological data analysis". In: *Neurocomputing* 596 (2024), p. 127787. ISSN: 0925-2312.

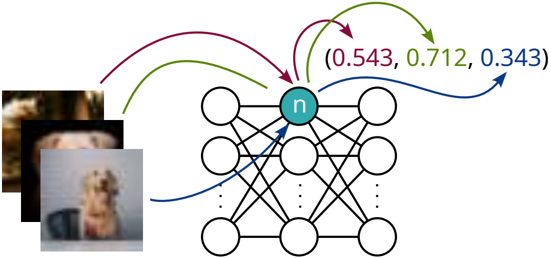
Generalization and persistent homology (I)

- **Objective**: Link the generalization gap of a neural network with the topological properties of the network and the training data. We follow Ballester et al. (2024).[1]
- **Hypothesis**: The persistent homology of the neuron activations of a neural network is connected to the generalization of the network.

[1] Rubén Ballester et al. "Predicting the generalization gap in neural networks using topological data analysis". In: *Neurocomputing* 596 (2024), p. 127787. ISSN: 0925-2312.

Generalization and persistent homology (I)

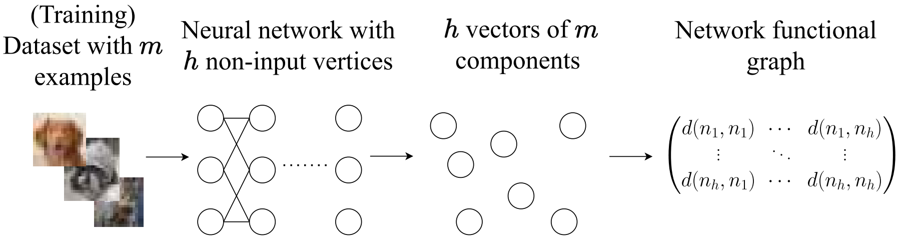
- **Objective**: Link the generalization gap of a neural network with the topological properties of the network and the training data. We follow Ballester et al. (2024).[1]
- **Hypothesis**: The persistent homology of the neuron activations of a neural network is connected to the generalization of the network.
- Given a neural network ϕ, a dataset $\mathcal{D}$, and a neuron n of ϕ, the **activation vector** of n is the vector $a_n = (\phi(x_1)_n, \ldots, \phi(x_m)_n)$, where $\phi_n(x)$ is the output of the neuron n for the input x.



[1] Rubén Ballester et al. "Predicting the generalization gap in neural networks using topological data analysis". In: *Neurocomputing* 596 (2024), p. 127787. ISSN: 0925-2312.

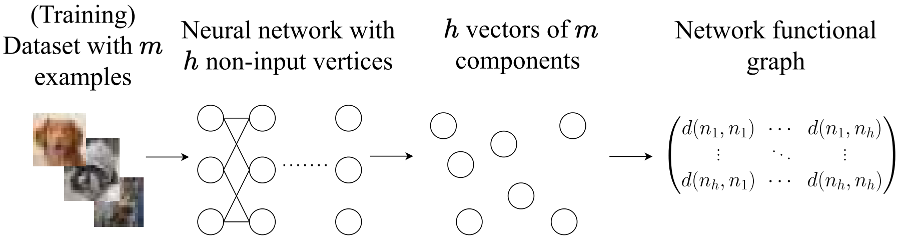
Generalization and persistent homology (II)

- After extracting all neuron activation vectors, we compute the persistent homology of the point cloud given by the non-input neurons using the dissimilarity $d(n_1, n_2) = 1 - |\mathrm{Corr}(a_{n_1}, a_{n_2})|$.



Generalization and persistent homology (II)

- After extracting all neuron activation vectors, we compute the persistent homology of the point cloud given by the non-input neurons using the dissimilarity $d(n_1, n_2) = 1 - |\mathrm{Corr}(a_{n_1}, a_{n_2})|$.

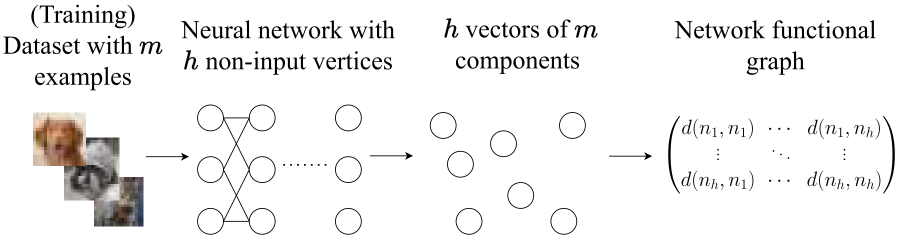


- Finally, we compare persistent homology vectorizations with the generalization of the network.

Generalization and persistent homology (II)

UNIVERSITAT DE BARCELONA

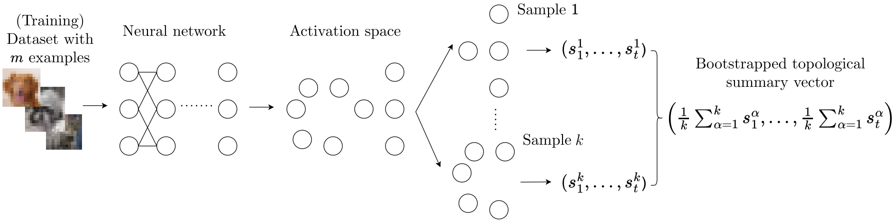
- After extracting all neuron activation vectors, we compute the persistent homology of the point cloud given by the non-input neurons using the dissimilarity $d(n_1, n_2) = 1 - |\mathrm{Corr}(a_{n_1}, a_{n_2})|$.



- Finally, we compare persistent homology vectorizations with the generalization of the network.
- Neural networks have thousands of neurons and thousands of samples. Is it *feasible* to compute the persistent homology of the activation vectors of all non-input neurons?

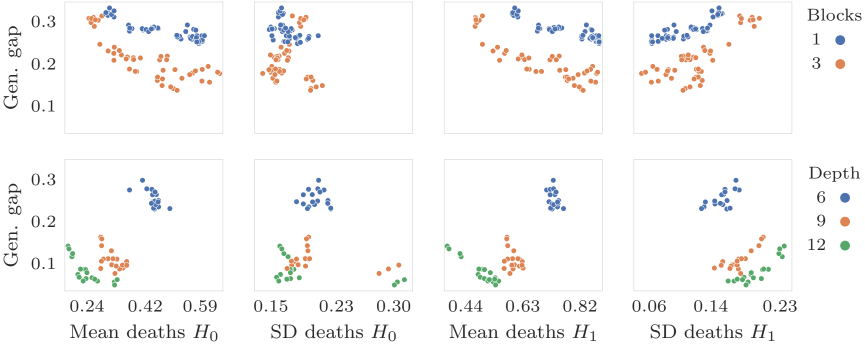
Generalization and persistent homology (III)

- The answer is **no**. We need to **sample** the dataset and the activation vectors of the neurons.
- To make the resulting vectorizations more robust, we use **bootstrap** methods.



Generalization and persistent homology (IV)

- We can now compare the persistent homology vectorizations with the generalization of the network on two sets[2] of different neural networks. Image from Ballester et al. (2024).



[2] Yiding Jiang et al. "Methods and Analysis of The First Competition in Predicting Generalization of Deep Learning". In: *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*. Ed. by Hugo Jair Escalante and Katja Hofmann. Vol. 133. Proceedings of Machine Learning Research. PMLR, June 2021.

Interpretability (I)

- **Objective**: Understanding how neural networks work internally. We follow the work done in the software TopoAct[3].

[3] Archit Rathore et al. "TopoAct: Visually Exploring the Shape of Activations in Deep Learning". In: *Computer Graphics Forum* (2021).

Interpretability (I)

- **Objective**: Understanding how neural networks work internally. We follow the work done in the software TopoAct[3].
- **Idea**: Understanding how data *activates* neurons can help us understand how data is processed by the network, and how the network makes decisions.

[3] Archit Rathore et al. "TopoAct: Visually Exploring the Shape of Activations in Deep Learning". In: *Computer Graphics Forum* (2021).

Interpretability (I)

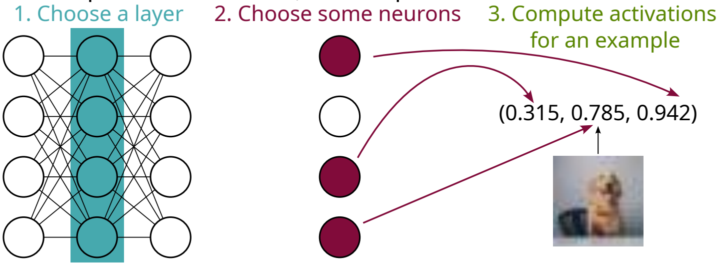
- **Objective**: Understanding how neural networks work internally. We follow the work done in the software TopoAct[3].
- **Idea**: Understanding how data *activates* neurons can help us understand how data is processed by the network, and how the network makes decisions.
- Mapper graphs built from neuron activations of a fixed layer help understanding how these activations are distributed according to the semantics of the data (e.g. classes).

[3] Archit Rathore et al. "TopoAct: Visually Exploring the Shape of Activations in Deep Learning". In: *Computer Graphics Forum* (2021).

Interpretability (II)

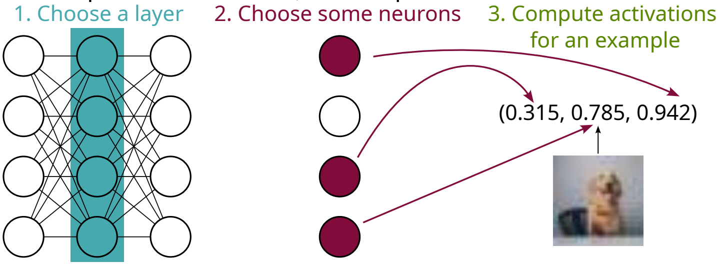
UNIVERSITAT DE BARCELONA

- Pipeline to compute the Mapper graph.
  1. For each input in a dataset $\mathcal{D}$, we compute an activation vector.



Interpretability (II)

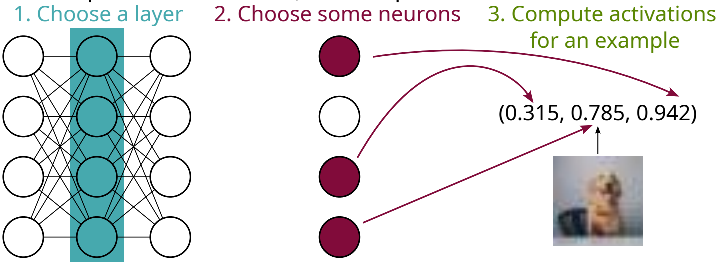
- Pipeline to compute the Mapper graph.
  1. For each input in a dataset $\mathcal{D}$, we compute an activation vector.



  2. Activation vector a_x for each $x \in \mathcal{D}$ form a point cloud with m points in $\mathbb{R}^d$, where d is the number of neurons selected.

Interpretability (II)

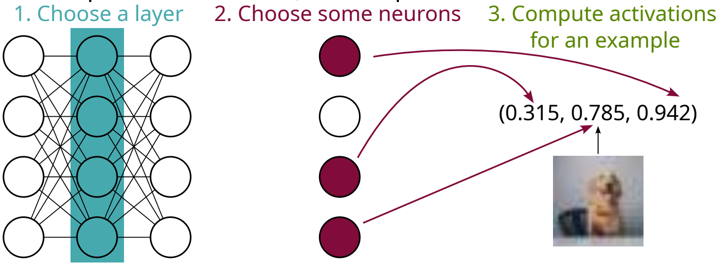
- Pipeline to compute the Mapper graph.
  1. For each input in a dataset $\mathcal{D}$, we compute an activation vector.



  2. Activation vector a_x for each $x \in \mathcal{D}$ form a point cloud with m points in $\mathbb{R}^d$, where d is the number of neurons selected.
  3. Generate a Mapper graph using $f(a_x) = \|a_x\|_2$, $\mathcal{U}$ a finite cover of uniformly sized overlapping intervals, and clustering algorithm `DBSCAN`.

Interpretability (II)

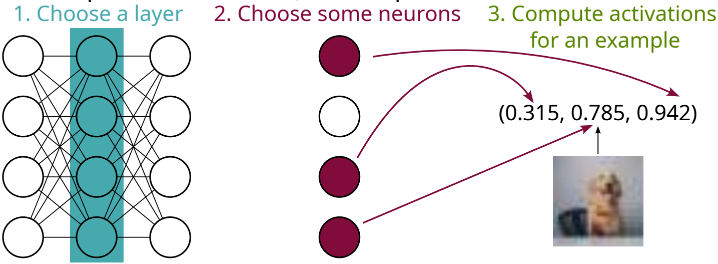
- Pipeline to compute the Mapper graph.
  1. For each input in a dataset $\mathcal{D}$, we compute an activation vector.



  2. Activation vector a_x for each $x \in \mathcal{D}$ form a point cloud with m points in $\mathbb{R}^d$, where d is the number of neurons selected.
  3. Generate a Mapper graph using $f(a_x) = \|a_x\|_2$, $\mathcal{U}$ a finite cover of uniformly sized overlapping intervals, and clustering algorithm DBSCAN.
- As each node in the Mapper graph represents a cluster of activation vectors, we can explore the images of each node and compute top labels, average activation, etc.

Interpretability (II)

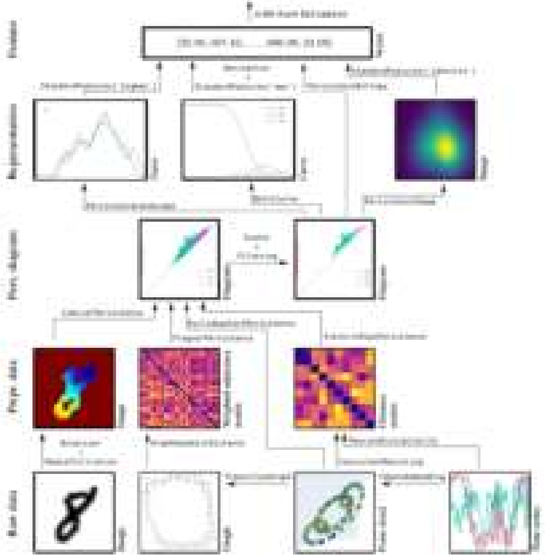
- Pipeline to compute the Mapper graph.
  1. For each input in a dataset $\mathcal{D}$, we compute an activation vector.



  2. Activation vector a_x for each $x \in \mathcal{D}$ form a point cloud with m points in $\mathbb{R}^d$, where d is the number of neurons selected.
  3. Generate a Mapper graph using $f(a_x) = \|a_x\|_2$, $\mathcal{U}$ a finite cover of uniformly sized overlapping intervals, and clustering algorithm DBSCAN.
- As each node in the Mapper graph represents a cluster of activation vectors, we can explore the images of each node and compute top labels, average activation, etc.
- Live demo: https://tdavislab.github.io/TopoAct/

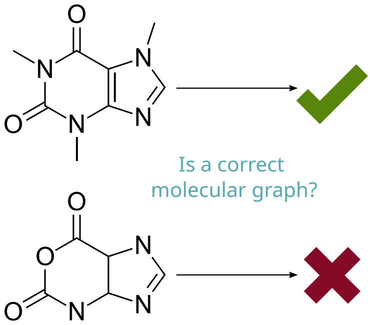
TDA as an input

- **Objective**: Use topological features of data as input for a network.



Persistent homology as a layer (I)

- Previously, we stated that the objective of deep learning is to approximate an unknown function $f\colon X \to Y$.
- What does it happen if X is not $\mathbb{R}^{d_i}$? For example, what does it happen if $X = \mathcal{G}$, the set of all finite graphs?



Persistent homology as a layer (II)

UNIVERSITAT DE BARCELONA

- **Idea**: Compute a vectorization of persistent homology of the input data as a layer of the network. One possible way to do this is to follow the PersLay approach[4].

[4] Mathieu Carriere et al. "PersLay: A Neural Network Layer for Persistence Diagrams and New Graph Topological Signatures". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 2786–2796. URL: https://proceedings.mlr.press/v108/carriere20a.html.

Persistent homology as a layer (II)

- **Idea**: Compute a vectorization of persistent homology of the input data as a layer of the network. One possible way to do this is to follow the PersLay approach[4].
- **PersLay** is a layer of the form

$$\mathrm{PersLay}(D) = \mathrm{op}\left(\{\{w(p)\phi(p)\}\}_{p\in D}\right),$$

where $w\colon \mathbb{R}^2 \to \mathbb{R}$ and $\phi\colon \mathbb{R}^2 \to \mathbb{R}^{d_i}$ are the weight and feature functions, respectively, and op is a permutation-invariant operation.

[4] Mathieu Carriere et al. "PersLay: A Neural Network Layer for Persistence Diagrams and New Graph Topological Signatures". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 2786–2796. URL: https://proceedings.mlr.press/v108/carriere20a.html.

Persistent homology as a layer (II)

- **Idea**: Compute a vectorization of persistent homology of the input data as a layer of the network. One possible way to do this is to follow the PersLay approach[4].
- **PersLay** is a layer of the form

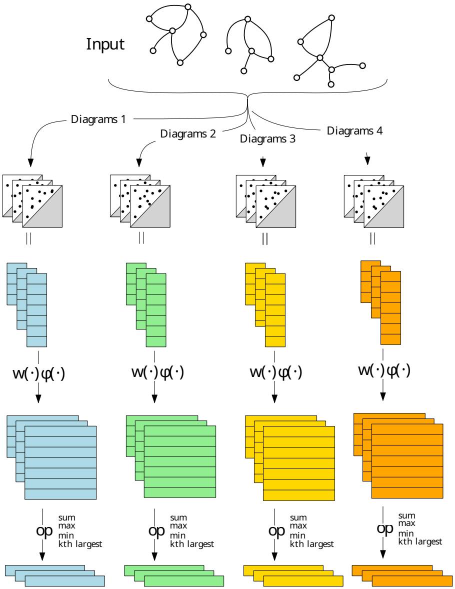
$$\mathrm{PersLay}(D) = \mathrm{op}\left(\{\{w(p)\phi(p)\}\}_{p\in D}\right),$$

where $w\colon \mathbb{R}^2 \to \mathbb{R}$ and $\phi\colon \mathbb{R}^2 \to \mathbb{R}^{d_i}$ are the weight and feature functions, respectively, and op is a permutation-invariant operation.
- Then, the classification problem is solved with the following composition:

$$(\mathrm{MLP} \circ \mathrm{PersLay} \circ \mathrm{Dgm}_k)(G).$$

[4] Mathieu Carriere et al. "PersLay: A Neural Network Layer for Persistence Diagrams and New Graph Topological Signatures". In: *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 26–28 Aug 2020, pp. 2786–2796. URL: https://proceedings.mlr.press/v108/carriere20a.html.

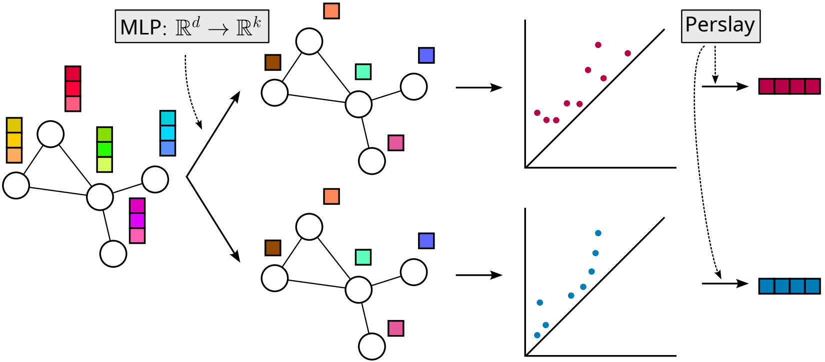
Persistent homology as a layer (III)



Learning filtrations (I)

Universitat de Barcelona

- Can we improve the previous pipeline? What if we learn the filtration?
- Many graph datasets have vectors $x_v \in \mathbb{R}^d$ associated to each node v.
- The filtration can be learned from these values using a neural network, as in Horn et al.(2022)[5].



[5] Max Horn et al. "Topological Graph Neural Networks". In: *International Conference on Learning Representations*. 2022.

Learning filtrations (II)

- One moment...

[6] Jacob Leygonie, Steve Oudot, and Ulrike Tillmann. “A Framework for Differential Calculus on Persistence Barcodes”. In: *Foundations of Computational Mathematics* 22.4 (Aug. 2022), pp. 1069–1131. ISSN: 1615-3383. DOI: 10.1007/s10208-021-09522-y. URL: https://doi.org/10.1007/s10208-021-09522-y.

Learning filtrations (II)

- One moment...
- In order to learn the weights of the MLP layer $\mathbb{R}^d \to \mathbb{R}^k$, it is necessary for the computation of persistent homology of the data to be differentiable.

[6] Jacob Leygonie, Steve Oudot, and Ulrike Tillmann. "A Framework for Differential Calculus on Persistence Barcodes". In: *Foundations of Computational Mathematics* 22.4 (Aug. 2022), pp. 1069–1131. ISSN: 1615-3383. DOI: 10.1007/s10208-021-09522-y. URL: https://doi.org/10.1007/s10208-021-09522-y.

Learning filtrations (II)

- One moment...
- In order to learn the weights of the MLP layer $\mathbb{R}^d \to \mathbb{R}^k$, it is necessary for the computation of persistent homology of the data to be differentiable.
- It turns out that the composition of the computation of the persistent homology and the PersLay layer is differentiable with respect to θ under some mild conditions.

[6] Jacob Leygonie, Steve Oudot, and Ulrike Tillmann. "A Framework for Differential Calculus on Persistence Barcodes". In: *Foundations of Computational Mathematics* 22.4 (Aug. 2022), pp. 1069–1131. ISSN: 1615-3383. DOI: 10.1007/s10208-021-09522-y. URL: https://doi.org/10.1007/s10208-021-09522-y.

Learning filtrations (II)

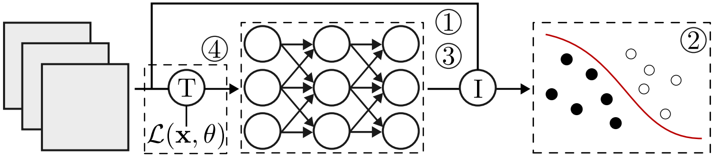
- One moment...
- In order to learn the weights of the MLP layer $\mathbb{R}^d \to \mathbb{R}^k$, it is necessary for the computation of persistent homology of the data to be differentiable.
- It turns out that the composition of the computation of the persistent homology and the PersLay layer is differentiable with respect to θ under some mild conditions.

More information about persistent homology and differentiability can be found in Leygonie et al. (2024)[6].

[6] Jacob Leygonie, Steve Oudot, and Ulrike Tillmann. "A Framework for Differential Calculus on Persistence Barcodes". In: *Foundations of Computational Mathematics* 22.4 (Aug. 2022), pp. 1069–1131. ISSN: 1615-3383. DOI: 10.1007/s10208-021-09522-y. URL: https://doi.org/10.1007/s10208-021-09522-y.

More on TDA for neural network analysis

Topological Data Analysis for Neural Network Analysis: A Comprehensive Survey



Conclusion

- Deep learning is a field of computer science focused on artificial intelligence by using neural networks.
- Neural networks are artificial models inspired by the brain's structure.
- Topological data analysis has been applied successfully in many areas of deep learning, including generalization, interpretability, input transformation, and architecture design.