Mapper applies the nerve construction to suitably customized coverings of datasets. Simplicial complexes produced by Mapper are often able to reveal patterns of a latent topological space from which the dataset might have been sampled. The Mapper algorithm has the following ingredients as input: 1. A dataset $\mathcal{D}$, which we assume of finite cardinality; 2. A filter function $f\colon \mathcal{D} \to \mathbb{R}^d$, where $d = 1$ or $d = 2$ in most use cases; 3. A finite cover $\mathcal{U} = \{\mathcal{U}_i\}_{i \in I}$ of $\operatorname{Im}(f) \subset \mathbb{R}^d$ consisting of overlapping open sets; 4. A clustering algorithm, such as `DBSCAN` (Ester et al., 1996). The method is described in Algorithm 1 in the Appendix.

Mapper is most often applied using filter functions $f\colon \mathcal{D} \to \mathbb{R}$ and choosing covers $\mathcal{U}$ consisting of intervals with a fixed percentage of overlap p. With these choices and using clustering algorithms that generate disjoint clusters, the simplicial complexes produced by Mapper are graphs. Mapper graphs are closely related with Reeb graphs (Reeb, 1946; Edelsbrunner and Harer, 2022, Chapter VI). For an advanced analysis of the Mapper algorithm for filter functions $f\colon \mathcal{D} \to \mathbb{R}$ and its connection with Reeb graphs, we refer the reader to the work by Carrière and Oudot (2018).

It was observed by Liu et al. (2023a) that, in some experiments studying the output space of neural networks, Mapper produced too many tiny components that did not carry useful information for the experiments. For this reason, they introduced a new algorithm called *graph-based topological data analysis* (GTDA). The GTDA algorithm builds on the Mapper algorithm to construct Reeb networks from graph inputs instead of point clouds. Reeb networks generalize Reeb graphs. Each vertex in a Reeb network built using GTDA is associated with a subgraph of the original graph. Given an input graph G, GTDA uses filter functions $f\colon V(G) \to \mathbb{R}^d$ to build Reeb network vertices using a recursive splitting strategy based on filter function values. After initial vertices are generated, the smallest ones are merged into other vertices and edges are added. Vertices within connected components with non-empty intersection are connected, and other edges are added to promote connectivity of the output. The GTDA algorithm is described in Liu et al. (2023a, Algorithm 1).

2.4 Applications of topological data analysis in deep learning

Some of the methods for analyzing neural networks in this survey have been applied to solve corresponding deep learning problems. In this section, we summarize and contextualize these deep learning problems.

2.4.1 Regularization

In machine learning, regularization techniques are *algorithmic tweaks* intended to reward models of lower complexity (Zhang et al., 2021). In the context of deep learning, regularization techniques aim to reduce model overfitting, that is, to avoid neural networks that excel in minimizing empirical risk $\widehat{\mathcal{R}}_{\mathcal{D}_{\text{train}}}$ but generalize poorly to a low real risk $\mathcal{R}$.

There are many ways in which regularization is performed in deep learning. Two of the most important ones are early stopping and the use of regularization terms.

In early stopping, training is stopped whenever a quality measure $\mathcal{Q}$, usually the empirical risk, evaluated on a set $\mathcal{D}_{\text{val}}$ different from the training dataset $\mathcal{D}_{\text{train}}$ stops improving. Defining the quality measure and the moment at which we consider that this measure has stopped improving is not an easy task, and it depends on the properties of the neural network and the training procedure. Some (non-topological) tips and tricks to tune early stopping can be found in Prechelt (1998).

Regularization terms are (almost everywhere) differentiable functions $\mathcal{T}\colon \Theta \to \mathbb{R}$ depending on the parameters of the neural network architecture being trained. They are minimized next to the empirical risk $\widehat{\mathcal{R}}_{\mathcal{D}_{\text{train}}}(\theta)$, i.e., making the training algorithm minimize the quantity

$$\widehat{\mathcal{R}}_{\mathcal{D}_{\text{train}}}(\theta) + \alpha\mathcal{T}(\theta),$$

with respect to the parameters θ, where $\alpha \in \mathbb{R}_{>0}$ is called *learning rate*, and controls the influence of the regularization term over the empirical risk.

2.4.2 Pruning of neural networks

Modern neural networks typically consist of highly complex architectures with a large number of parameters. Working with such models usually requires large amounts of computational resources that may not be easily available or sustainable in the long term. One of the approaches to solve this problem is neural network pruning. Neural network pruning involves systematically removing parameters from an existing network so that the resulting network after this process conserves as much performance as possible, but with a drastically reduced number of parameters (Blalock et al., 2020).

2.4.3 Detection of adversarial, out-of-distribution, and shifted examples

Adversarial, out-of-distribution, and shifted examples are specific values in the input space of a neural network that have special properties with respect to the underlying distribution of the input data or the way the neural network processes them. Given a machine learning model, adversarial examples are inputs that differ slightly from correctly classified inputs but are misclassified by the model. Differences between adversarial examples and properly classified regular examples are often imperceptible to human senses (Goodfellow et al., 2015). On the other hand, out-of-distribution examples are valid input values for a machine learning model that have not been sampled from the real data distribution. Similarly, shifted examples are examples that deviate from the real data distribution but come from samples from it, resembling the nature of adversarial examples (Lacombe et al., 2021).

Detecting input types is an important problem in deep learning. On one hand, it allows further study of the robustness and performance of a given neural network. On the other hand, it allows to detect attacks to machine learning models, both in training and in inference.

2.4.4 Detection of trojaned networks

In a Trojan attack on a machine learning model (Zheng et al., 2021), attackers create *trojaned* examples, that are normal training examples to which some extra information, using specific patterns, is added. This extra information is used by attackers to generate specific outputs in the inputs containing these specific patterns, allowing attackers to bypass the regular outputs of the model during inference. To be able to generate these *unwanted* outputs, attackers inject trojaned examples into the original training data set without the knowledge of the legitimate owner of the model before the model is trained with the objective that the network recognizes the pattern and acts according to the attackers' intention. Detecting neural networks trained with trojaned examples is vital to avoid security breaches in machine learning, especially in critical-context applications such as medical systems or security systems, among others.

2.4.5 Model selection

Over the last years, there has been an explosion on the quantity and quality of models developed by deep learning researchers and practitioners. Most of these models have a large number of parameters and are hard and expensive to train. However, once trained successfully, they generalize well to many situations. For this reason, a reasonable practice in nowadays applications is, instead of training from zero new models, taking pre-trained models and adapting them to the particular problem the user is trying to solve. However, selecting the pre-trained model that best fits the user's requirements from a *bank* of models is not a trivial task. For starters, each bank of models has its own models, trained in different datasets with different algorithms and different hardware. Small variations of the parameters in the same architecture can induce important differences during inference, so selecting only by the architecture is not usually the best option. Additionally, due to the growing number of options available, testing all of them may be an unfeasible process for practical applications. For this reason, a good model selection algorithm that takes into account many properties of the models is fundamental.

2.4.6 Prediction of accuracy

In classification tasks, a usual loss function is given by

$$\mathcal{L}(\mathcal{N}, x, y) = \mathbb{1}_{\{(x,y)\,:\,x\neq y\}}\left(\mathcal{N}(x), y\right). \tag{9}$$

Assuming that the values of $\mathcal{Y}$ are fully determined by a function $f\colon \mathcal{X} \to \mathcal{Y}$ in the data distribution, the real risk $\mathcal{R}(\mathcal{N})$ for this loss function is the expectation that the neural network $\mathcal{N}$ does not match the function f for an input value x, that is, misclassifies x. If instead we consider the value $1 - \mathcal{R}(\mathcal{N})$, this is the expectation that the neural network correctly classifies an input x, i.e., the probability of $\mathcal{N}$ matching f. We define this value by accuracy and denote it by $\mathrm{Acc}(\mathcal{N}) = 1 - \mathcal{R}(\mathcal{N})$. The higher the accuracy, the better the model according to the risk of $\mathcal{L}$. The problem with $\mathrm{Acc}(\mathcal{N})$ is that we cannot compute it, so we approximate it by the empirical accuracy of $\mathcal{N}$ in a sample $\mathcal{D}$ different from the training dataset $\mathcal{D}_{\text{train}}$, usually called test dataset and denoted by $\mathcal{D}_{\text{test}}$. Therefore, this empirical accuracy in $\mathcal{D}$ can be computed as

$$\widehat{\mathrm{Acc}}_{\mathcal{D}}(\mathcal{N}) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} \mathbb{1}_{\{(x,y)\,:\,x=y\}}\left(\mathcal{N}(x), y\right). \tag{10}$$

The empirical accuracy on a test dataset $\mathcal{D}_{\text{test}}$ is one of the most used metrics to assess the generalization and performance of a trained neural network and is equivalent to studying a numerical approximation of the risk function and thus, the generalization capacity of the neural network. Due to the hardness of proving tight bounds on the real risk $\mathcal{R}$ of a neural network given $\mathcal{L}$, many works focus on predicting the empirical accuracy $\widehat{\mathrm{Acc}}_{\mathcal{D}_{\text{test}}}(\mathcal{N})$ or a