PersLay: A Neural Network Layer for Persistence Diagrams and New Graph Topological Signatures (Summary)

University of Utrecht

Audrique Vertessen 2377764, Georgios Smyridis 1840118

May 30, 2022

This is a summary of a paper[1]. This is thus fully based (up to some changes) on this paper. We will not keep citing this paper through out this summary but this should be kept in the back of your mind.

Introduction

It is know that using persistent homology can give insights in a dataset. However it can be hard to interpret the persistent diagrams if the application of persistent homology results in many persistent diagrams. When facing such a problem one would like to apply standard machine learning techniques to the set of persistent diagrams. In this paper techniques are developed to achieve this goal. There are two main parts to solve this problem. The first one is converting persistent diagrams into finite vectors, which is needed to apply standard machine learning techniques. The second part of the problem is the learning part itself. This is mainly done in two ways. Either with support vector machines (SVD) or with neural networks. In this paper neural networks are chosen. By doing this one would need to developed a layer in the neural network that would work for the chosen vectorization of the persistent diagrams and that can easily be implemented and complement existing neural network architectures. The layer that was developed in this paper is called PersLay. Then, the developed techniques were applied to classification on large scale dynamical system dataset and classification of graphs. Finally we also applied this to classify stock data (this was not done in the paper).

Theory

Extended Persistence

We start this section of by first introducing the notion of extended persistence for graphs[1] , which as the name suggest, is a generalisation of the ordinary persistence. With extended persistence one is able to define death of loops and whole connected components. Also more detailed topological features will be able to be detected. First, we need to define some other notions.

Definition 1 *Let $G = (V, E)$ be a graph with vertex set V and the set of (non-oriented) edges E. Also let $f : V \to \mathbb{R}$ be a function defined on the vertex set of G. The sublevel graphs are defined by $G_\alpha = (V_\alpha, E_\alpha)$ with $\alpha \in \mathbb{R}$, $V_\alpha = \{v \in V : f(v) \leq \alpha\}$ and $E_\alpha = \{(v_1, v_2) \in E : v_1, v_2 \in V_\alpha\}$.*

Definition 2 *Let the set-up be such as in definition 1. The superlevel graphs are defined by $G^\alpha = (V^\alpha, E^\alpha)$ with $\alpha \in \mathbb{R}$, $V^\alpha = \{v \in V : f(v) \geq \alpha\}$ and $E^\alpha = \{(v_1, v_2) \in E : v_1, v_2 \in V^\alpha\}$.*

With these definitions we can extend the notion of death times for topological features of graphs. We can do this by keeping the original notion of birth/death of a feature but extend the notion of death of a feature to also be when it first appears again in the superlevel graphs. Lets say that this superlevel graph is G^{α_d} then, α_d would be defined as the death of that feature.

Definition 3 *An extended persistence diagram of (G, f), $Dg(G, f) \subset \mathbb{R}$,is the multiset of births and deaths intervals, $[\alpha_b, \alpha_d]$ formed by the sublevel and superlevel graphs of G and then using the generalised notion of death.*

[1]This can be defined for general metric spaces.

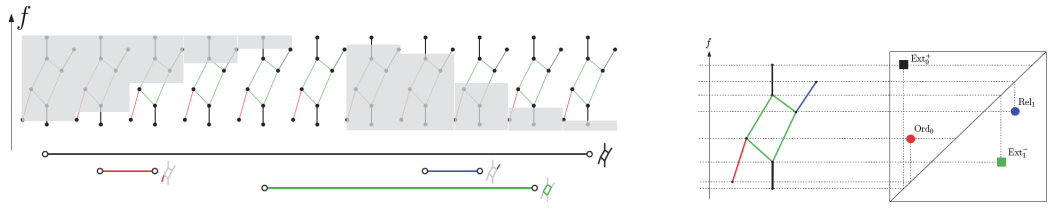

Figure 1: Example of extended persistence.

Heat kernel signatures

The goal of this section is to find a function to build extended persistent diagrams out of (see definition 3) such that certain properties are satisfied. We start of by defining some short definitions. Consider a graph G with vertex set $V = \{v_1, ..., v_n\}$. $A_{ij} = (\mathbb{1}_{(v_i,v_j)\in E})_{ij}$ is the adjacency matrix of G. $D_{ii} = \sum_{j=1}^{n} A_{ij}$ is the degree matrix. The matrix $L_w = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ is called the normalized graph Laplacian. This matrix acts on the space of functions defined on V. An orthonormal basis of eigenfunctions $\Psi = \{\psi_1, ..., \psi_n\}$ with eigenvalues satisfying $0 \leq \lambda_1 \leq ... \leq \lambda_n \leq 2$ can be found for L_w. Of course this basis is not uniquely defined and can thus not be used to compare graphs. Instead we use the Heat Kernel Signature (HKS).

Definition 4 *Given a graph G and $t \geq 0$, the Heat Kernel Signature with diffusion parameter t is the function, $hks_{G,t}$ defined on the vertices of G by $hks_{G,t} : v \to \sum_{k=1}^{n} exp(-t\lambda_k)\psi_k(v)^2$.*

Earlier it was mentioned that we wanted to define a function that could be used to build extended persistence diagrams but also satisfied certain properties. The following two theorems are the desired properties.

Theorem 1 ***Stability w.r.t. graph perturbations.*** *Let $t \geq 0$ and let L_w be the normalized graph Laplacian matrix of a graph G with n vertices. Let G' be another graph with n vertices and Laplacian matrix $\tilde{L}_w = L_w + W$. Then there exists a constant $C(G,t) > 0$ only depending on t and the spectrum of L_w such that, for small enough $||W||_F$, where $||.||_F$ denotes the Frobenius norm and $d_B(.)$ denotes the bottleneck distance:*

$$d_B(Dg(G,t), Dg(G',t)) \leq C(G,t)||W||_F \tag{1}$$

For the next theorem we first state that the map $t \to Dg(G,t)$ is Lipschitz-continuous.

Theorem 2 ***Stability w.r.t. diffusion parameter t.*** *Let G be a graph. The map $t \to Dg(G,t)$ is 2-Lipschitz continuous, that is for $t, t' \in \mathbb{R}$,*

$$d_B(Dg(G,t), Dg(G,t')) \leq 2|t - t'| \tag{2}$$

Neural network learning with PersLay

In this section we introduce PersLay, a neural network layer for processing topological information encoded in persistence diagrams. We note that it is possible to generate a neural architecture layer for persistence diagrams by composing any neural network architecture ρ with PersLay.

Definition 5 (PersLay) *PersLay is a generic neural network layer for persistence diagrams defined by the following equation:*

$$\textit{PersLay}(Dg) := \boldsymbol{op}(\{w(p) \cdot \phi(p)\}_{p \in Dg}) \tag{3}$$

where ***op*** *is any permutation invariant operation (such as minimum, maximum, sum kth largest value etc.),* $w : \mathbb{R}^2 \to \mathbb{R}$ *is a weight function for the persistence diagram points, and* $\phi : \mathbb{R}^2 \to \mathbb{R}^q$ *is a representation function that we call point transformation, mapping each point* (α_b, α_d) *of a persistence diagram to vector.*

We now introduce three point transformations that are can used to implement Eq.(3):

- The *triangle point transformation* $\phi_\Lambda : \mathbb{R}^2 \to \mathbb{R}^q$, $p \mapsto [\Lambda_p(t_1), \Lambda_p(t_2), \ldots, \Lambda_p(t_q)]^T$ where the triangle function Λ_p associated to a point $p = (x, y) \in \mathbb{R}^2$ is $\Lambda_p : t \mapsto \max\{0, y - |t - x|\}$ with $q \in \mathbb{N}$ and $t_1, \ldots, t_q \in \mathbb{R}$.
- The *Gaussian point transformation* $\phi_\Gamma : \mathbb{R}^2 \to \mathbb{R}^q$, $p \mapsto [\Gamma_p(t_1), \Gamma_p(t_2), \ldots, \Gamma_p(t_q)]^T$, where the Gaussian function Γ_p associated to a point $(x, y) \in \mathbb{R}^2$ is $\Gamma_p : t \mapsto \exp(-||p - t||_2^2/(2\sigma^2))$ for a given $\sigma > 0$, $q \in \mathbb{N}$ and $t_1, \ldots, t_q \in \mathbb{R}^2$
- The *line point transformation* $\phi_L : \mathbb{R}^2 \to \mathbb{R}^q$, $p \mapsto [L_{\Delta_1}(p), L_{\Delta_2}(p), \ldots, L_{\Delta_q}(p)]$, where the line function L_Δ associated to a line Δ with direction vector $e_\Delta \in \mathbb{R}^2$ and bias $b_\Delta \in \mathbb{R}$ is $L_\Delta : p \mapsto \langle p, e_\Delta \rangle + b_\Delta$, with $q \in \mathbb{N}$ and $\Delta_1, \Delta_2, \ldots, \Delta_q$ are q lines in the plane.

Even though PersLay is simple, it is highly versatile. By taking different combinations of permutation invariant operations **op** and point transformation functions ϕ, PersLay can remarkably encode most of classical persistence diagram vectorisations. For this reason, ϕ can be considered as a hyperparameter of sorts. Some examples are the following:

- Using $\phi = \phi_\Lambda$ with samples $t_1, \ldots, t_q \in \mathbb{R}$, **op** = kth largest value, $w = 1$ (a constant weight function), amounts to evaluating the kth *persistence landscape* on $t_1, \ldots, t_q \in \mathbb{R}$.
- Using $\phi = \phi_\Lambda$ with samples $t_1, \ldots, t_q \in \mathbb{R}$, **op**=sum, arbitrary weight function w, amounts to evaluating the *persistence silhouette* weighted by w on $t_1, \ldots, t_q \in \mathbb{R}$.
- Using $\phi = \phi_\Gamma$ with samples $t_1, \ldots, t_q \in \mathbb{R}^2$, **op** = sum, arbitrary weight function w, amounts to evaluating the *persistence surface* weighted by w on $t_1, \ldots, t_q \in \mathbb{R}^2$. Moreover, characterizing points of persistence diagrams with Gaussian functions is also the approach advocated in several kernel methods for persistence diagrams.

Stability of PersLay

In TDA, the issue of the continuity and stability of persistence diagram vectorisations is of importance. The operation defined in Eq.(3) – with **op** = sum – is not continuous in general, at least with respect to common persistence diagram metrics. Actually, for $s \geq 1$ the map $\mathrm{Dg} \mapsto \sum_{p \in \mathrm{Dg}} \phi(p)$ is continuous with respect to d_s if and only if ϕ is of the form $\phi(p) = \varphi(p)||p - \Delta||^s$, where $||p - \Delta||$ is the distance from a point $p \in \mathbb{R}^2$ to the diagonal $\Delta = \{(x, x), x \in \mathbb{R}\}$ and φ is a continuous and bounded function.

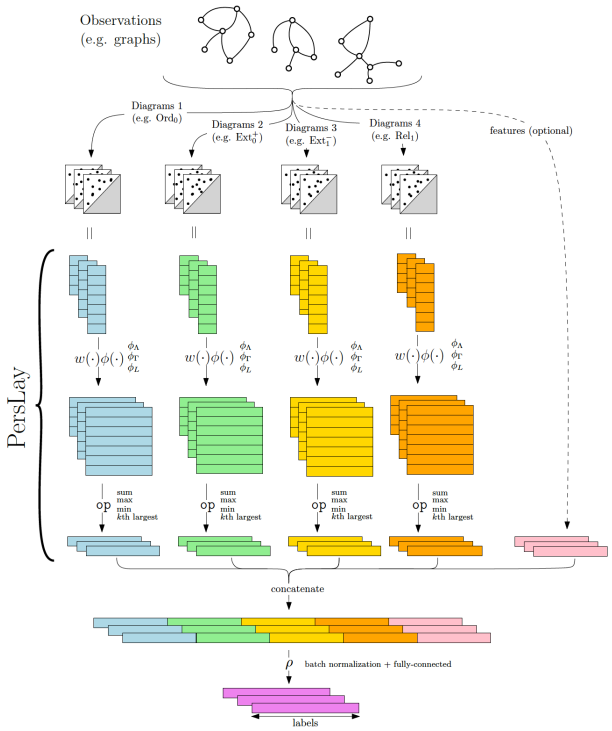

Figure 2: An example of the PersLay architecture used in graph classification.

Applications

After we have reviewed the theory behind the extended persistence and the PersLay neural network layer, we would like to implement these tools on actual datasets. For that purpose we use the PersLay class, functions and methods as well as benchmark datasets provided by the authors available at https://github.com/MathieuCarriere/perslay.

For each dataset, we perform a 10 ten-fold evaluations and report the average and best ten-fold results. For a ten-fold evaluation, we split the data into ten equally-sized parts and record the classification accuracy obtained by the on the ith fold (test) after training the network with the nine remaining folds.

Graph Classification

To begin with, we use the extended persistence introduced in previous section to generate the diagrams that encode the topological features of the graph datasets under study. Then, we use the PersLay neural network layer to classify the graphs based on said topological information. Specifically, we apply the aforementioned on two different benchmark graph datasets, namely "MUTAG" and "PROTEINS". There is a certain freedom to choosing the different hyperparameters of the PersLay layer. At the beginning, we use the same as the authors in order to compare our results. We can verify that the results we get are comparable to the ones the authors provide in the paper

(see Figure), while any deviation most probably originates in the random choices our code makes, like random splitting of datasets into folds etc.

	PersLay	SV	RetGK	FHSD	GCNN	GIN
Dataset	Mean	–	–	–	–	–
MUTAG	83.7	88.3	90.3	92.1	86.7	89.0
PROTEINS	72.9	72.6	75.8	73.4	76.3	75.9
NCI1	65.0	71.6	84.5	79.8	78.4	82.7

Figure 3: Mean classification accuracy over benchmark labeled graph datasets by PersLay as well as other state–of–the–art graph classification techniques.

Remarkably, even thought the neural network layer architecture we are using is very simple, the classification accuracy is comparable to state of the art results. However, for the NCI1 dataset, both topology-based methods (SV and PersLay) had pretty mediocre performances, which suggests that topology cannot discriminate and thus classify graphs in this dataset.

Variation of Hyperparameters

Then, we vary the values of the hyperparameters of PersLay and see how the accuracy of the classification changes. We did this for the MUTAG dataset since this one was by far the smallest. Such hyperparameters are all the things that can be changed in definition 3 or if after the PersLay-layer we apply another neural network architecture (post processing).

	PersLay
Changes w.r.t. settings of figure 3	Mean
No post processing	86.3
op= *mean*	83.7
Smaller grid weight function	86.8
Landscape layer without post processing	88.4
Landscape layer, no post processing and power weight function	78.4

Figure 4: Mean classification accuracy on the graph dataset MUTAG by PersLay with different hyperparameter settings.

Classification of large scale dynamical system datasets

We now move on and use the PersLay layer in order to classify the orbits of large scale dynamical systems. The orbits of the dynamical systems we study are generated in the following way: Given some initial position $(x_0, y_0) \in [0,1]^2$ and a parameter $r > 0$, we generate a point cloud

$(x_n, y_n)_{n=1,\dots,N}$ by:

$$\begin{cases} x_{n+1} = x_n + r y_n(1 - y_n) \mod 1 \\ y_{n+1} = y_n + r x_{n+1}(1 - x_{n+1}) \mod 1 \end{cases} \tag{4}$$

The orbits of this dynamical system depend heavily on the parameter r. As one can see in figure 6 for some values of r (of course it also depends on the initial position), voids might form in these orbits and thus, persistence diagrams are likely to be effective in classifying the orbits with respect to the value of the parameter r. The parameters we choose for r are $\{2.5, 3.5, 4.0, 4.1, 4.3\}$ and for each one we generate an orbit of $N = 1,000$ points and thus, we get the dataset "ORBIT5K". Then these point clouds are turned into persistence diagrams using the AlphaComplex filtration in dimensions 0 and 1. We split the observations in 70-30 (%) training-test sets and report the average accuracy over 100 runs, like the authors did. The accuracy results are the following: We can see

Dataset	PSS-K	PWG–K	SW–K	PF–K	PersLay (Mean)
ORBIT5K	72.38	76.63	83.6	85.9	87.2

Figure 5: Perslay classification performance for **OBRITS5K** dataset compared to state of the art methods.

again that the performance of PersLay is comparable to state–of–the–art classification methods.

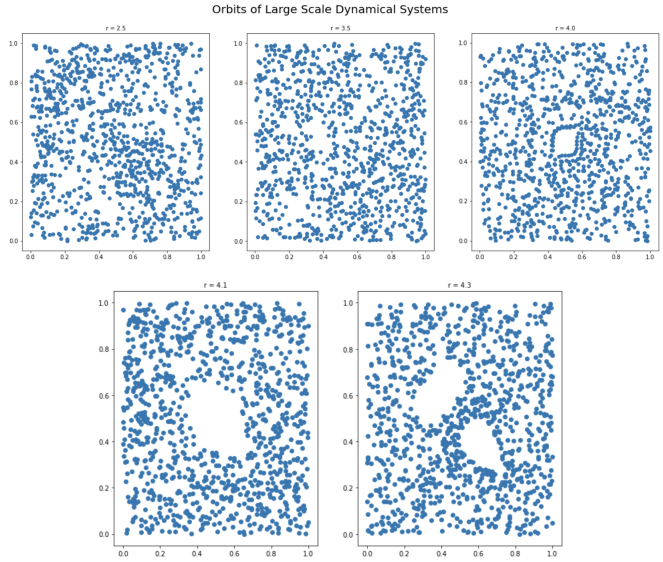

Figure 6: 5K orbits for different values of the parameter r

One can generate the dataset "ORBIT100K" by choosing each orbit to contain 20,000 points, however we did not because the execution time increases significantly. According to the authors however, this increase in the number of points per orbit leads to an increase in classification accuracy.

Classification applied to stock market time-series

Finally we applied this machinery in a new context. We classified a day (of a stock) to be a 'buy' if the next day was at least 1% higher, a 'sell' if it was at least 1% lower and 'do nothing' for the rest. We will call this '1%' the cutoff since we can change this (this is a kind of hyperparameter). We used 1 year (∼250 days of data) of minute stock data. For each day (∼650 points) we used Takens Embedding Theorem (in 3 dimensions) and then generated the persistent diagrams for 0, 1 and 2 dimensional homology.
We used the same settings as the authors used in their analysis of the dataset MUTAG. There was not enough time to vary these. We were however able to test this setting with extra features (not only persistent diagrams) (see figure 2), on two different stocks (SPY and NVDA) and with a different cutoff value.

	SPY	NVDA
Changes w.r.t. original setup	Mean	Mean
None	64.4	32.0
feature: closing price of the day	70.8	36.8
feature: daily interest rates	72.4	38.4
feature: all minute data	5.6	23.2
Cutoff = 0.03	–	69.6

Figure 7: Mean classification accuracy stocks SPY and NVDA by PersLay with features and cutoff values.

A few interesting things can be seen figure 7. First note that a cutoff of 1% works decently well for SPY but doesn't work at all for NVDA. But a 3% cutoff does work well for NVDA (couldn't be tested on SPY since there were only a few days with more than 3% swings). A possible reason could be that for SPY 'non-random' swings are of the order of more 1% but since NVDA is naturally more volatile the 'non-random' swings have higher values (∼3%). Another thing that could be seen is that for the extra features of closing price and interest rates the accuracy improved a little bit. A somewhat unexpected thing is that if we include all the minute data as features then the accuracy is totally ruined. We are not really sure why this is the case but it could be that there is to much random movement included in the minute data such that the neural network is trained on mostly random data. This could mean that in the best case the neural network doesn't learn anything and in the worst case that it learns the wrong things.

Conclusions

In this paper they managed to generate stable extended persistent diagrams for graphs using the heat kernel signature of that graph. After that PersLay was defined, which is a neural network

layer for persistent diagrams. PersLay is very versatile. By changing the hyperparameters it is possible to reproduce most of the well know vectorisations such as the persistence surface. The final part consisted of applications of PersLay. PersLay performed comparable to the state–of–the–art classification methods in graph classification and classification of large scale dynamical systems. The impact of changing the hyperparameters of PersLay was also explored briefly. In a later project these could also be learned. A new application of PersLay was also explored. Namely classifying stock data. The results were decent but should definitly be explored further.

Bibliography

[1] Mathieu Carrière, Frédéric Chazal, Yuichi Ike, Théo Lacombe, Martin Royer, and Yuhei Umeda. Perslay: A neural network layer for persistence diagrams and new graph topological signatures, 2020.