"*This looks like something I am confident about*"
ProtoWNet - Weight prediction with prototype similarity

Gabriele Dominici

Abstract

Neural Networks are very powerful tools, but it is difficult to explain their reasoning process. However, we can exploit related examples to support the decision a system makes on a particular instance. Similarly, I developed a model called *prototypical weighted network* (ProtoWNet), which makes the decisions by combining the prediction of the learnt prototypes using their similarity with the sample we are treating. It makes the model more interpretable and allows it to learn the right amount of prototypes. In addition to that, it achieves comparable performance or even better than the black box model.

1. Introduction

Neural Networks are becoming increasingly present in our daily life, but their unclear reasoning process and lack of explanation represent a huge problem. Therefore, using interpretable models in some real-life situations is essential due to legal aspects and to ensure a more conscious use of them (Doshi-Velez & Kim, 2017).

For these reasons, several solutions have been proposed over the last decade to increase the explainability and interpretability of these models. Some of them are post-hoc methods that highlight some parts of the input to justify the prediction (Ribeiro et al., 2016; Lundberg & Lee, 2017; Selvaraju et al., 2019), while others are self-explainable models which, with little modification, greatly improve the clarity of their reasoning (Koh et al., 2020; Alvarez-Melis & Jaakkola, 2018). One such model was presented by Chen et al. (2018) (ProtoPNet). The model uses the similarity between the learnt prototypes and the sample we want to classify to make the prediction. The main idea was to mimic a human reasoning process that we apply when we support our decisions using examples that we have seen previously.

Inspired by this model, I developed the *prototypical weighted network* (ProtoWNet). It uses the same underlying idea of ProtoPNet but applies it slightly differently. First, it selects a set of training samples and uses them as prototypes. Then it computes the prediction for these prototypes and the similarity of their representation with each of the other samples we want to classify. Finally, to compute the final prediction for a sample, the model calculates the weighted sum of the prediction of the prototypes that we computed previously, using the similarities between that sample and the prototypes as weights.

This new solution has several key points. Firstly, this solution uses samples in the training set as prototypes without learning hypothetical representations and projecting them. Secondly, it can use and learn a variable number of prototypes without defining that number in advance because the weighted sum can deal with a variable number of prototypes. This implies that this new solution can learn a number of prototypes which is sufficient to solve the task. Moreover, the decision part does not rely on any Neural Network, making the decision process clearer.

Moreover, I propose a new metric, Prototype Accuracy (ProtoAcc), to evaluate the prototypes used to support the predictions quantitatively. It computes the model's accuracy using the label of the most similar prototype.

I tested ProtoWNet on a few Node Classification datasets, applying it at the end of a Graph Convolutional Network (GCN) (Kipf & Welling, 2016). In this field, there are some datasets where the idea of prototypes fits perfectly, and it is easy to show them. However, this approach can be applied to other modalities. The results on these datasets show that ProtoWNet achieves comparable or, in some cases, better results than the Vanilla GCN, making the model more interpretable. In addition, the ProtoAcc achieved by the model is similar to the raw accuracy, proving that the explanations given by the most similar prototypes are valuable.

2. Background

ProtoPNet (Chen et al., 2018) was the first model to introduce prototypes inside the model. It proposed to justify the decision with the similarity of a sample with the learnt prototypes. A prototype, in this case, is a learnt embedding in the latent space, representing a peculiar example of a specific class and can be visualised as the closest example in the training set. Specifically, the model is trained to learn a specific number of prototypes per class and then is trained

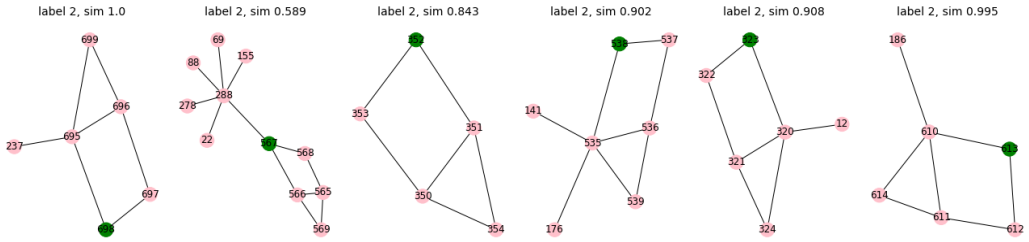

Figure 1. Explanation step of ProtoWNet on a BAShapes node. The first example is the node we want to classify, and the others are the top five prototypes with the highest similarity score with respect to that input. The node is coloured green, and the nodes in the 2-hop neighbourhood are coloured rose. As you can see, the input has the same structural importance as the two most similar prototypes. Moreover, the other prototypes with high similarity are nodes from the same class with a slightly different structures.

to predict the right labels given the similarity score between prototypes and samples. In addition, it proposed two extra loss terms to learn prototypes clustered with samples of the same class and well separated from the samples of the other classes.

The original paper applied it to image datasets, but Zhang et al. (2021) proposed a slightly different solution to apply it to graphs. The idea was the same with an additional module to find the closer subgraph to the prototypes learnt.

Other popular ways to make GNN models more interpretable are GNNExplainer(Ying et al., 2019), GCExplainer (Magister et al., 2021) and Concept Encoder Module (CEM)(Magister et al., 2022). The first one tries to learn a subgraph that has a minimal difference between the model's prediction of it and on the initial graph. It aims to select the most important part of the graph for predictions. It can be plugged after the model we want to explain, and it should be trained to predict the minimal subgraph. On the other hand, GCExplainer and CEM utilise concepts to explain GNN architectures. GCExplainer is a post-hoc method which uses KMeans to cluster the latent representation of graphs and nodes before the prediction step, discovering some concepts. Similarly, CEM learns concepts but in a fully differentiable fashion.

3. ProtoWNet

ProtoWNet maintains the idea proposed by ProtoPNet (Chen et al., 2018) of supporting the reasoning process by reference to specific examples that we have seen previously, but it proposes a different way to select prototypes, choosing the right amount of them needed to solve the task. Section 3.1 explains how to select prototypes.

The ProtoWNet can be applied before the prediction layer. First, it takes the latent representations of the samples, and, after normalising them to unit vectors (norm equals 1), it computes the prediction for the chosen prototypes ($\hat{\mathbf{Y}}_\mathbf{p} \in \mathbf{R}^{\mathbf{k}\times\mathbf{c}}$ where $\mathbf{k}$ is the number of prototypes and $\mathbf{c}$ the number of classes). Subsequently, it calculates the cosine similarity between each sample and each prototype ($\mathbf{Sim} \in \mathbf{R}^{\mathbf{n}\times\mathbf{k}}$ where $\mathbf{n}$ is the number of samples). At this stage, it computes the weighted sum of the prototype predictions $\hat{\mathbf{Y}}_\mathbf{p}$ using $\mathbf{Sim}$ as weights. The last step is to apply the LogSoftmax to output the logarithm of the probabilities ($\hat{\mathbf{Y}} \in \mathbf{R}^{\mathbf{n}\times\mathbf{c}}$).

$$\hat{\mathbf{Y}} = LogSoftmax(\mathbf{Sim}\hat{\mathbf{Y}}_\mathbf{p})$$

In contrast to ProtoPNet, this solution does not use any Deep Neural Network (DNN) to make the final prediction, making the reasoning process clearer. To explain the prediction of an input, you can visualise the prototypes and show how much each contributes to it, as shown in Figure 1. Moreover, as you can see from the definition of the two matrices, the number of prototypes can be a variable number that can change from iteration to iteration, not constraining the model to use a limited number of them per class.

Figure 2 shows the backbone of ProtoWNet applied on a Graph Convolutional Network (Kipf & Welling, 2016).

3.1. Selection of prototype

The selection of prototypes is a key process of the model because it impacts the final prediction process. It is possible to design it in different ways. For instance, I used a random selection (Section 3.1.1) as a baseline and a more sophisticated one that selects the most diverse samples with higher prediction confidence as prototypes. The first solution uses a fixed number of prototypes per class, while the second one decides how many to use.

3.1.1. Random prototype

The easiest way to select some prototypes in the training set is to randomly choose $\mathbf{k}$ prototypes per class. This is not optimal but represents a good baseline to use. Using this approach, the chosen prototype could not represent

the whole set of inputs, and there is no guarantee that two prototypes represent different things.

3.1.2. Diverse and Confident Prototype Selection (DCPS)

The other method I have designed chooses the prototypes by selecting the most diverse samples with the higher prediction confidence per class. For instance, it takes the top percentile (**top p**) of samples with the highest predictions on a specific class, and iteratively it excludes samples that have a similarity score greater than a threshold (**cos threshold**) with one of the others that have a higher prediction. It repeats this process for each class. Changing these two hyperparameters, you can specify how many samples you are considering when you choose the prototype and how much they should be diverse.

This allows you to specify some conditions you want to be respected when choosing the prototypes rather than specify a fixed number. This allows the model to use the right amount of prototypes that meet these prerequisites to solve the task.

In this way, the model pursues the "this looks like that" philosophy proposed by Chen et al. (2018), but it extends it to "this looks like something that I am confident about".

However, it is possible to structure the training process in two different ways.

Sequential You train the model with respect to the task you want to solve, and when it converges, you select the prototypes as previously explained. Then, you freeze all layers before the one that makes the prediction. By doing this, the model will have learned how to represent each sample according to the task. From that point forward, the similarity between samples and prototypes will remain constant. At this step, the model only tweaks the last layer used to compute the predictions for the prototypes.

Joint Starting from the first epoch, you train the model choosing the best prototypes at each iteration. In this way, the model learns how to solve the task directly using the prototypes that it is choosing. The prototype chosen at each step influences the latent representations learnt by this model.

3.2. Loss function

Another important aspect of the model is the loss function used. As proposed in ProtoPNet (Chen et al., 2018), I implemented the cluster cost (*Clst*) and the separation cost (*Sep*). The first one forces the model to have at least a prototype with high similarity for each sample of the same class. On the other hand, the second one encourages the model to have low similarity between each sample and prototypes of other classes.

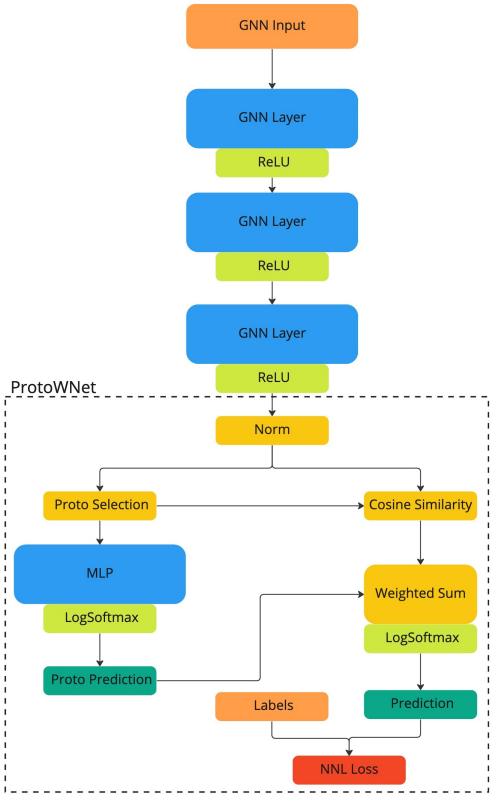

Figure 2. ProtoWNet applied on a Graph Convolutional Network.

These two terms are added to the NLL Loss. The complete loss can be summarised with the following equations:

$$Clst = -\frac{1}{n}\sum_{i=1}^{n} \max_{j:y_{pj}=y_i} \mathbf{Sim}_{ij}$$

$$Sep = \frac{1}{n}\sum_{i=1}^{n} \max_{j:y_{pj}\neq y_i} \mathbf{Sim}_{ij}$$

$$L = NLL(\mathbf{Y}, \hat{\mathbf{Y}}) + \alpha Clst + \beta Sep$$

where α and β are hyperparameters.

The sequential approach only uses prototypes once it converges, but it still hypotheses to choose them at every iteration and computes all the similarity scores, using them in the $Clst$ and Sep terms of the loss.

4. Experiments

I tested the proposed model in multiple Node Classification tasks and tried various combinations of the introduced hy-

perparameters. Specifically, I explored different values of α and β to assess their impact on the performance and the discovered prototypes. Additionally, I employed diverse values of $\mathbf{k}$ in the random selection to evaluate their effect on the outcomes. For the same reason, I also used different values of ($\mathbf{top\ p}$) and ($\mathbf{cos\ threshold}$) in the DCPS method.

However, I did not do hyperparameter tuning on other variables like learning rate, number of epochs or depth and width of the networks used. All the further details about the parameters used are described in Appendix A.

4.1. Datasets

I used three synthetic and one real-world Node Classification datasets. The synthetic ones were designed by (Ying et al., 2019) to support the explainability property of their model. I have chosen these datasets because they are structured so that it is easy to show if a prototype represents a node well. They are composed of specific motifs, visually recognizable, that we know in advance.

BAShapes The graph is a Barabasi-Albert graph with additional five-node house-structured network motifs attached to random nodes of it. In addition to that, some random edges were added to create more noise. There are 4 labels: one for nodes belonging to the Barabasi-Albert graph, one for the top nodes of the house motif, one for the middle nodes and the last for bottom nodes. The initial node embedding is the same value across the whole graph.

BAGrid The graph is a Barabasi-Albert graph with additional 3-by-3 grid network motifs attached to random nodes of it. In addition to that, some random edges were added to create more noise. There are 2 labels: one for nodes that belong to the Barabasi-Albert graph and the other for the nodes that belong to the grid. The initial node embedding is the same value across the whole graph.

BACommunity The graph is the union of two BAShapes graphs. Each of the two BAShapes graphs is considered a community. In this case, the labels are eight and are based on the structural roles of the node, as in BAShapes, and on the community membership. The initial node embedding is a normally distributed vector, where the distribution's mean and variance are specific to the community.

Cora It is the citation graph of thousand scientific publications. Each node represents a publication, and each edge is a citation between two research papers. Each node embedding is the bag of words of the text of the paper, while its label is its topic. There are seven different classes.

Each dataset was split into training and test sets with an 80-20 split.

4.2. Number of prototypes

I did several experiments to understand how the number of prototypes impacts the results achieved by the model. For the random selection of prototypes, I have tested four different $\mathbf{k}$ (2, 4, 8, 16), which represents the number of prototypes per class. It is possible to enforce the model to select more prototypes also in the DCPS approach reducing the similarity threshold ($\mathbf{cos\ threshold}$) and the top percentile considered by the model when it looks for prototypes ($\mathbf{top\ p}$). Specifically, I tried three $\mathbf{cos\ threshold}$ (0.98, 0.95, 0.90) and three $\mathbf{top\ p}$ (0.01, 0.05, 0.1).

4.3. Loss parameters

Other hyperparameters of the model are the α and β parameters of the loss function. In my experiments, I used $\alpha = \beta$ to reduce the number of experiments and balance the loss between these two terms. I tested four factors (0, 0.2, 0.5, 1). These experiments aim to inspect how these hyperparameters influence the performance of the model and the prototypes found.

4.4. Goodness of prototype learnt

In previous works (Chen et al., 2018; Zhang et al., 2021), the authors evaluated their models only with the accuracy on the given tasks, but there was no automatic metric to evaluate the prototypes found by the models.

For this reason, I proposed a measure (Prototype Accuracy - ProtoAcc) to understand if the prototypes found match with classified samples. It is the model's accuracy if, to classify a sample, we use the label of the most similar prototype with it. It mimics the *completeness score* used in concept-based models.

It cannot measure the real similarity of two inputs, but at least it identifies if the most similar prototype shares the label with the input.

5. Results

Table 1 shows that ProtoWNet can provide explanations while still achieving performance similar to black box models. It can do it with different configurations, both with a random selection of prototypes and using the DPCS method. Additionally, the loss factor is not helpful in all configurations and datasets.

However, I omitted all the results on the BAGrid dataset because they were less informative, as all the tested models performed similarly, achieving an almost perfect score.

Model	BAShapes		BACommunity		Cora	
	Acc	ProtoAcc	Acc	ProtoAcc	Acc	ProtoAcc
Vanilla GCN	**0.97** ±0.02	-	0.84 ±0.02	-	0.85 ±0.01	-
Random $\alpha = 0$	0.95 ±0.01	0.95 ±0.01	0.84 ±0.03	0.83 ±0.03	0.82 ±0.02	0.82 ±0.02
Random $\alpha = 0.2$	0.95 ±0.01	0.95 ±0.02	0.87 ±0.03	0.83 ±0.02	0.83 ±0.02	**0.86** ±0.02
Sequential $\alpha = 0$	**0.97** ±0.01	**0.96** ±0.01	**0.88** ±0.03	**0.88** ±0.02	0.83 ±0.02	0.84 ±0.02
Sequential $\alpha = 0.2$	0.91 ±0.10	0.81 ±0.12	0.85 ±0.02	0.85 ±0.02	**0.86** ±0.01	**0.86** ±0.02
Joint $\alpha = 0$	0.88 ±0.06	0.86 ±0.08	0.87 ±0.02	0.85 ±0.01	0.85 ±0.02	0.85 ±0.02
Joint $\alpha = 0.2$	0.96 ±0.02	0.90 ±0.05	0.83 ±0.03	0.76 ±0.02	0.83 ±0.02	0.82 ±0.02

Table 1. Results for different configurations of ProtoWNet, on different datasets. It can be compared with the Vanilla GCN, which is a black box model. The Random approaches used a $\mathbf{k} = 8$, the Sequential ones a $\mathbf{top\ p} = 0.05$ and a $\mathbf{cos\ threshold} = 0.95$, while the Joint ones a $\mathbf{top\ p} = 0.10$ and a $\mathbf{cos\ threshold} = 0.95$. Every model was initialised with 5 different seeds (0, 8, 13, 24, 42).

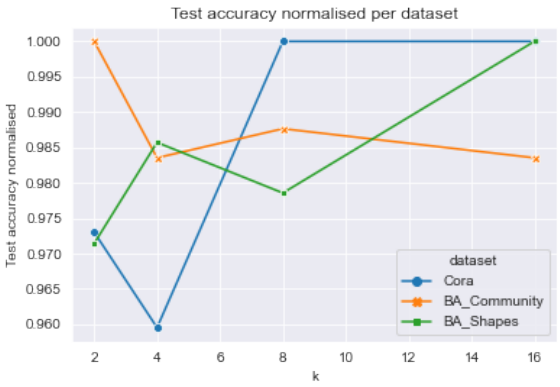

Figure 3. Normalised test accuracy of random selection models with different $\mathbf{k}$ on all datasets. It is divided by the highest accuracy achieved in that dataset.

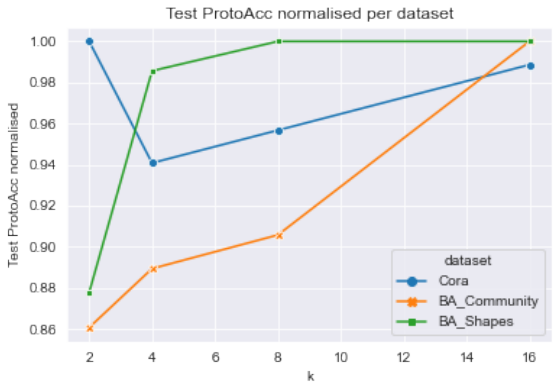

Figure 4. Normalised test ProtoAcc of random selection models with different $\mathbf{k}$ on all datasets. It is divided by the highest ProtoAcc achieved in that dataset.

5.1. Prototype selection

How the model selects prototypes is crucial to both the level of explainability and the model's performance, as they consistently influence its predictions.

However, Table 1 shows that even with a random selection of $\mathbf{k}$ prototypes per class, with a $\mathbf{k}$ sufficiently high, it can achieve good accuracy and ProtoAcc. It does not mean that the top-ranked prototype matches perfectly with the input but that they share the same label. Moreover, we have no certainty that the prototypes used are sufficiently different. These results confirm that justifying the usefulness of prototype models (ProtoPNet and ProtoGNN) by using only the accuracy is misleading.

On the other hand, the Sequential approach achieved the best results in all datasets, with even slightly better accuracy than the black box model. Moreover, its ProtoAcc is only marginally lower than the accuracy, proving that the explanations given share the same label. In contrast to random selection, with this approach, we have defined how much two prototypes should be different to be selected, ideally improving the explanations given and reducing the number of useless prototypes. This solution learns the best latent representations to solve the task and then changes the prediction step, making the training process more stable. In addition to that, it is possible to apply this method to pre-trained models, changing the prediction layer. This is equivalent to using $\alpha = 0$ and $\beta = 0$.

To use as a comparison, GNNExplainer (Ying et al., 2019) achieves an explanation accuracy, which is the equivalent of the ProtoAcc, of 0.83 on BACommunity and 0.93 on BAShapes. GCExplainer (Magister et al., 2021) reached 0.68 and 0.96, respectively, in completeness score.

Lastly, the joint solution tries to learn the best prototypes to use as described in Section 3.1.2 while it learns how to classify them. It can change the chosen prototypes at each iteration, modifying the input expected by the prototype prediction layer. It could increase the difficulty of the model in clustering samples accordingly to the prototypes. This

Figure 5. Number of prototypes chosen by the sequential and joint model over time on the BACommunity dataset.

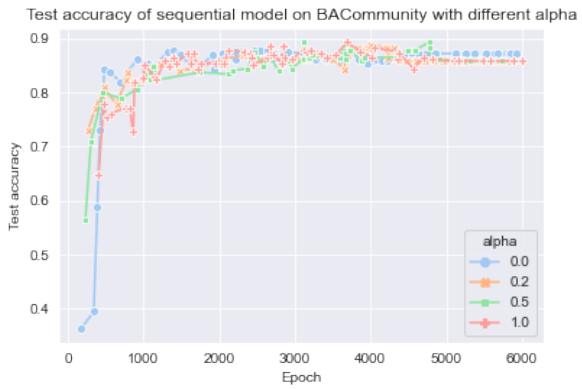

Figure 6. Test accuracy of the sequential model on the BACommunity dataset.

makes the training process more unstable resulting in lower and more variant results. This also causes worse performances on the ProtoAcc.

5.2. Number of prototypes

Then I examined how the number of prototypes found impacted the model results and how the number of prototypes chosen by the DCPS method change over time.

Figure 3 shows the results obtained in different datasets by the random selection model changing the number of prototypes per class. The accuracy is normalised with the highest result in that dataset, making comparing the results on different datasets easier. Overall, it seems that using more prototypes helps the model achieve better performance, which is reasonable because it increases the chance of picking prototypes that better represent the input space. Figure 4 shows clear evidence that more prototypes imply higher ProtoAcc, and it happens for the same reason. However, this increase is not linear, and after a threshold on the number of prototypes, the performance stops increasing. However, having more prototypes increases the model's inference time, as it has to compute $n\epsilon$ more cosine similarity, where n is the number of samples, and ϵ is the number of extra prototypes.

On the other hand, it is interesting to see what happens in the Sequential approach. Figure 5 shows how the number of prototypes varies and the model's accuracy over the training process. It is visible how, after some epochs where the number increase substantially, it starts to drop, although the accuracy remains stable. This could be imagined as an automated pruning of useless prototypes. However, the joint model acts oppositely. The number of prototypes increases and then remains stable over time.

In the last two methods, changing $\mathbf{top\ p}$ and $\mathbf{cos\ threshold}$ modifies the number of prototypes found by the model, but there is no clear metrics improvement using more prototypes. It could happen because they are already filtered and are the most meaningful prototypes. Sometimes, you do not need the extra fine-grained prototypes discovered with higher $\mathbf{cos\ threshold}$ or lower $\mathbf{top\ p}$ to solve a task.

5.3. Loss factor

The experiments reveal that the difference between using a small α (0.2) and a bigger one (1.0) is irrelevant. For example, Figure 6 shows what happened in the BACommunity datasets using the Sequential approach, but almost the same happened on all the other datasets with all the different approaches.

From the performance point of view, there is no clear answer if it is better to use it or not. Table 1 shows that in some cases, it is helpful, especially in the random selection approach, while in the other two approaches, it depends on the task.

5.4. ProtoAcc

As shown in previous sections, ProtoAcc can be a good metric to measure if the prototypes proposed are in some sense related to the input. However, even the random selection process can achieve moderately good scores, even if the selected prototypes cannot faithfully represent all samples. Therefore, other factors should be considered to have a broader view of the prototypes found. For instance, on Node Classification tasks, it could be helpful to measure how much different the structure of the neighbourhood of a node is with respect to the neighbourhood of the best prototype. The same could also be calculated among different prototypes of the same class to measure if the selected proto-

types are structurally diverse and not only different enough in the latent space.

Further experiments can be done measuring these two metrics using the edit distance between two subgraphs. It is computed as the number of actions to transform one graph into another. They can be calculated, respectively, in the following way:

$$TopEDist = \frac{1}{n}\sum_{x=1}^{n}\frac{1}{EditDist(\mathbf{N}_x, \mathbf{N}_p)+1}$$

where n is the number of samples, p is the prototype with the highest similarity with respect to x and $\mathbf{N}_x$ is the subgraph of the neighbourhood of the node x.

$$ProtoEDist = 1 - \frac{1}{p^2}\sum_{i=1}^{p}\sum_{j=1}^{p}\frac{1}{EditDist(\mathbf{N}_i, \mathbf{N}_j)+1}$$

6. Conclusions

In this project, I proposed a new model, ProtoWNet, that uses prototypes to justify its predictions. In contrast to ProtoPNet, it does not use any DNNs to do the last prediction step but computes them, combining the prototypes' predictions using the similarity of the prototypes with each sample. Moreover, the proposed model is not constrained to learn a specific number of prototypes per class, while it can learn prototypes with a specific level of granularity. I also proposed ProtoAcc, a new metric to evaluate if the top-ranked prototype for each sample is related to it.

I tested this solution on several Node Classification tasks. On all of them, the model achieves similar results in accuracy to the vanilla model in all configurations, but it also provides examples to support predictions. Moreover, it reached similar figures also for the ProtoAcc, showing that the proposed prototypes are meaningful. However, more metrics should be introduced to understand the prototypes' quality fully.

Despite having tested several configurations of the model proposed, the final ones can still be improved. There is no unique configuration that works better in all situations. It depends on the task and the user's needs. Additional experiments can be done by testing additional combinations of parameters and finetuning more general hyperparameters like the learning rate and the number of epochs. Moreover, other methods to select prototypes can be explored in further analysis, providing, for instance, more stability to the joint training process or pruning useless prototypes.

Furthermore, it would be interesting to implement this model on other GNN layers and in different modalities, comparing it to ProtoPNet on image datasets, for example.

All the experiments can be reproduced with the code available in a GitHub repository[1].

References

Alvarez-Melis, D. and Jaakkola, T. S. Towards robust interpretability with self-explaining neural networks. 2018. doi: 10.48550/ARXIV.1806.07538. URL https://arxiv.org/abs/1806.07538.

Chen, C., Li, O., Tao, C., Barnett, A. J., Su, J., and Rudin, C. This looks like that: Deep learning for interpretable image recognition. 2018. doi: 10.48550/ARXIV.1806.10574. URL https://arxiv.org/abs/1806.10574.

Doshi-Velez, F. and Kim, B. Towards a rigorous science of interpretable machine learning. 2017. doi: 10.48550/ARXIV.1702.08608. URL https://arxiv.org/abs/1702.08608.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. 2016. doi: 10.48550/ARXIV.1609.02907. URL https://arxiv.org/abs/1609.02907.

Koh, P. W., Nguyen, T., Tang, Y. S., Mussmann, S., Pierson, E., Kim, B., and Liang, P. Concept bottleneck models. 2020. doi: 10.48550/ARXIV.2007.04612. URL https://arxiv.org/abs/2007.04612.

Lundberg, S. and Lee, S.-I. A unified approach to interpreting model predictions. 2017. doi: 10.48550/ARXIV.1705.07874. URL https://arxiv.org/abs/1705.07874.

Magister, L. C., Kazhdan, D., Singh, V., and Liò, P. Gcexplainer: Human-in-the-loop concept-based explanations for graph neural networks. 2021. doi: 10.48550/ARXIV.2107.11889. URL https://arxiv.org/abs/2107.11889.

Magister, L. C., Barbiero, P., Kazhdan, D., Siciliano, F., Ciravegna, G., Silvestri, F., Jamnik, M., and Lio, P. Encoding concepts in graph neural networks. 2022. doi: 10.48550/ARXIV.2207.13586. URL https://arxiv.org/abs/2207.13586.

Ribeiro, M. T., Singh, S., and Guestrin, C. "why should i trust you?": Explaining the predictions of any classifier. 2016. doi: 10.48550/ARXIV.1602.04938. URL https://arxiv.org/abs/1602.04938.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. Grad-CAM: Visual explanations from deep networks via gradient-based

[1] https://github.com/gabriele-dominici/R255_XAI_project

localization. *International Journal of Computer Vision*, 128(2):336–359, oct 2019. doi: 10.1007/s11263-019-01228-7. URL https://doi.org/10.1007%2Fs11263-019-01228-7.

Ying, R., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. Gnnexplainer: Generating explanations for graph neural networks. 2019. doi: 10.48550/ARXIV.1903.03894. URL https://arxiv.org/abs/1903.03894.

Zhang, Z., Liu, Q., Wang, H., Lu, C., and Lee, C. Protgnn: Towards self-explaining graph neural networks. 2021. doi: 10.48550/ARXIV.2112.00911. URL https://arxiv.org/abs/2112.00911.

A. Hyperparametrs used

Table 2 shows the other hyperparameters used to train the models used in the experiments.

B. Example of prototypes

Figures 7, 8, 9, 10 show the two prototypes for each class that are ranked first in similarity more times found by sequential model trained on BAShapes, BAGrid, BACommunity and Cora, respectively.

Dataset	Learning rate	Number of layer	Hidden dimension
BAShapes	0.001	4	20
BAGrid	0.001	4	20
BACommunity	0.001	6	30
Cora	0.001	2	20

Table 2. Hyperparams used to train every models on a specific dataset.

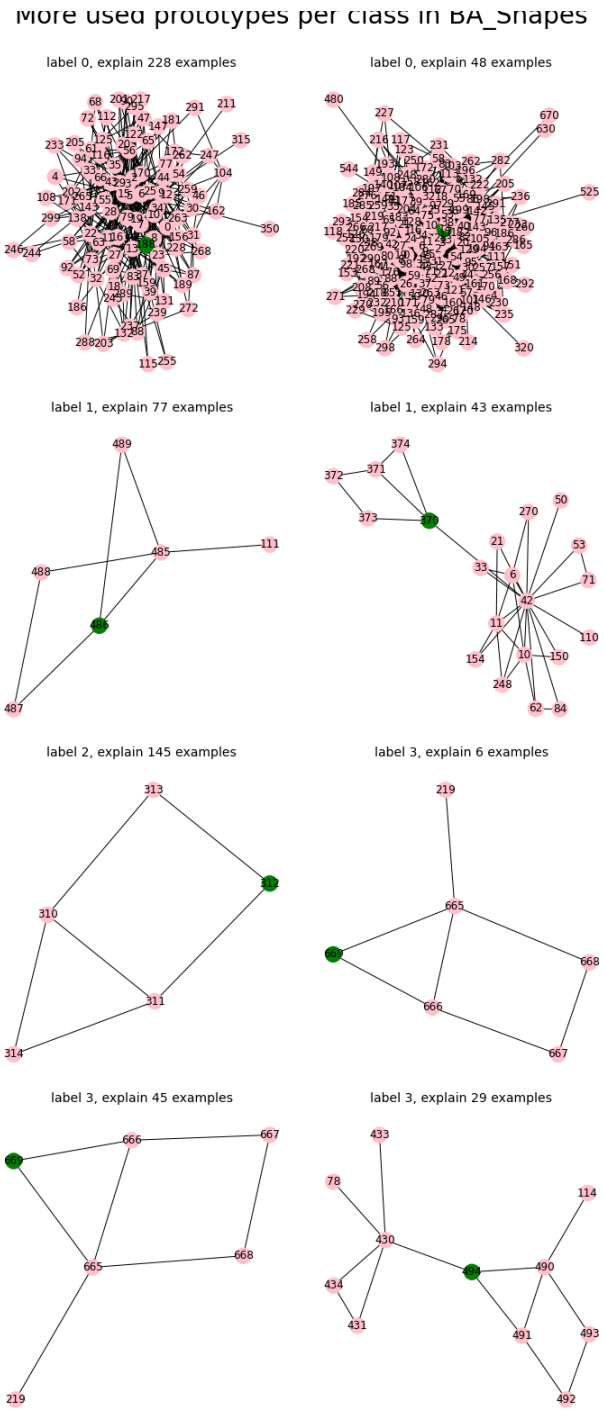

Figure 7. The two prototypes which are ranked first in similarity with samples of a class more times. Each row is a class. Each prototype shows its label and how many samples of the class represented by the row have it as the most similar prototype. They are the prototypes found by the sequential model trained on BAShapes.

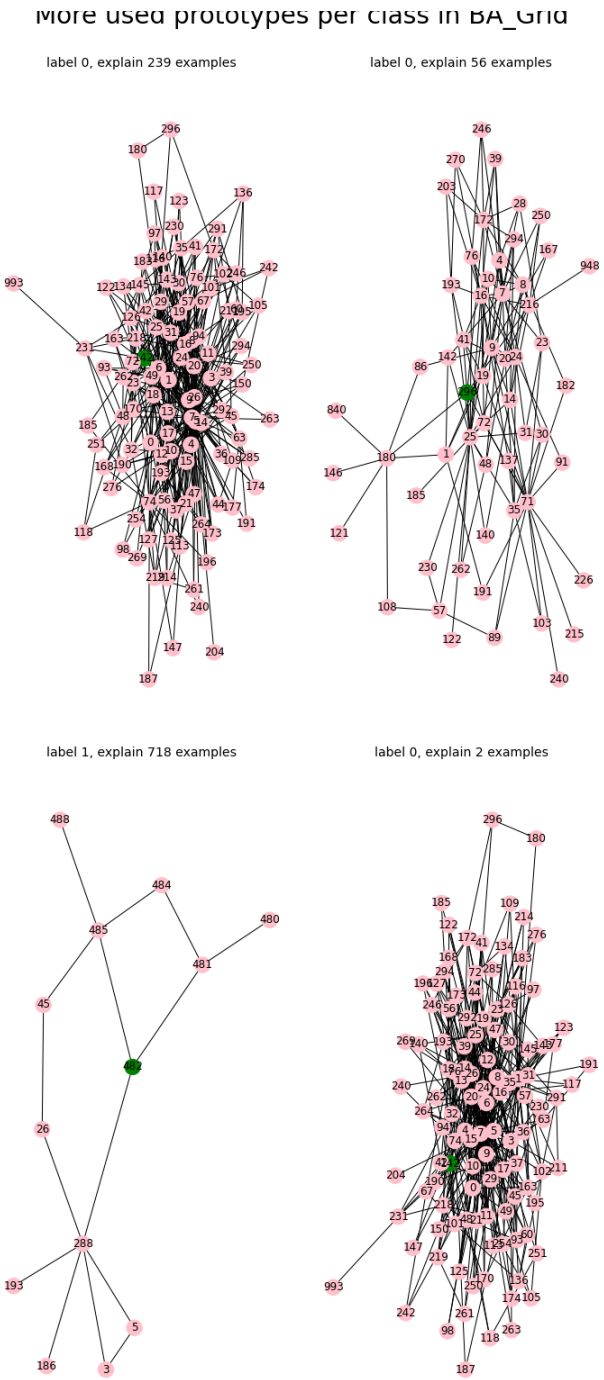

Figure 8. The two prototypes which are ranked first in similarity with samples of a class more times. Each row is a class. Each prototype shows its label and how many samples of the class represented by the row have it as the most similar prototype. They are the prototypes found by the sequential model trained on BAGrid.

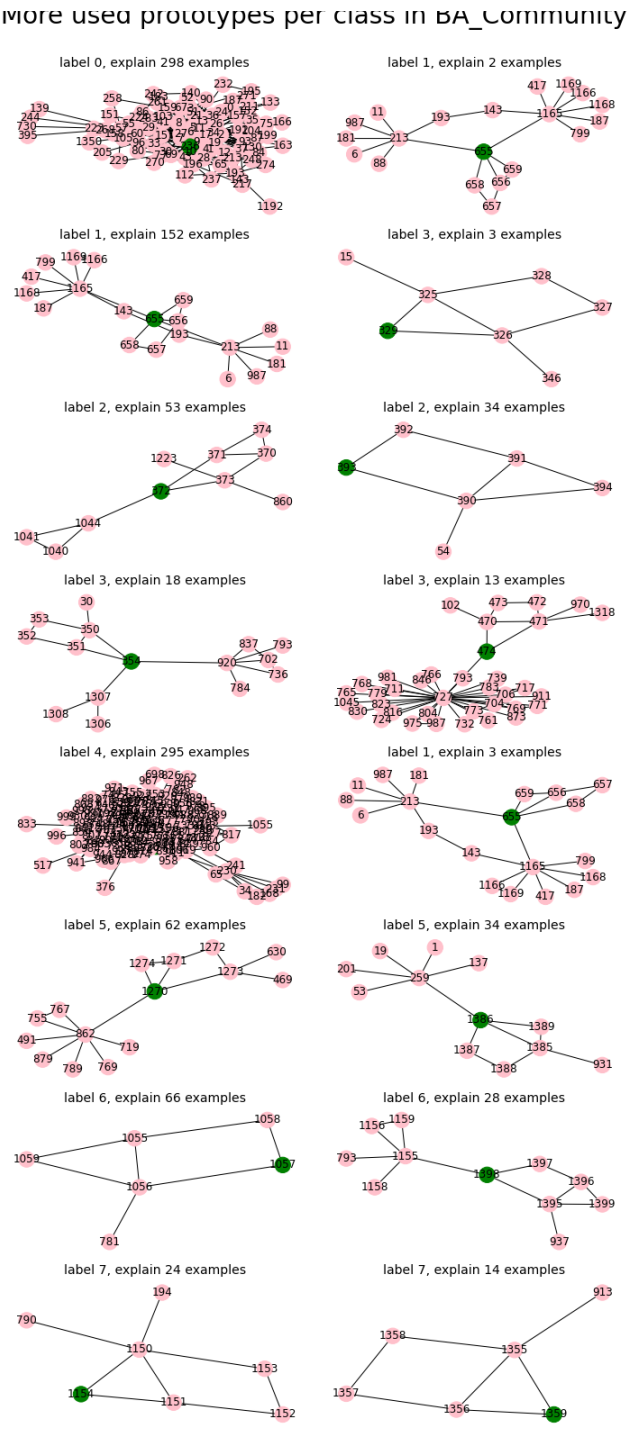

Figure 9. The two prototypes which are ranked first in similarity with samples of a class more times. Each row is a class. Each prototype shows its label and how many samples of the class represented by the row have it as the most similar prototype. They are the prototypes found by the sequential model trained on BACommunity.

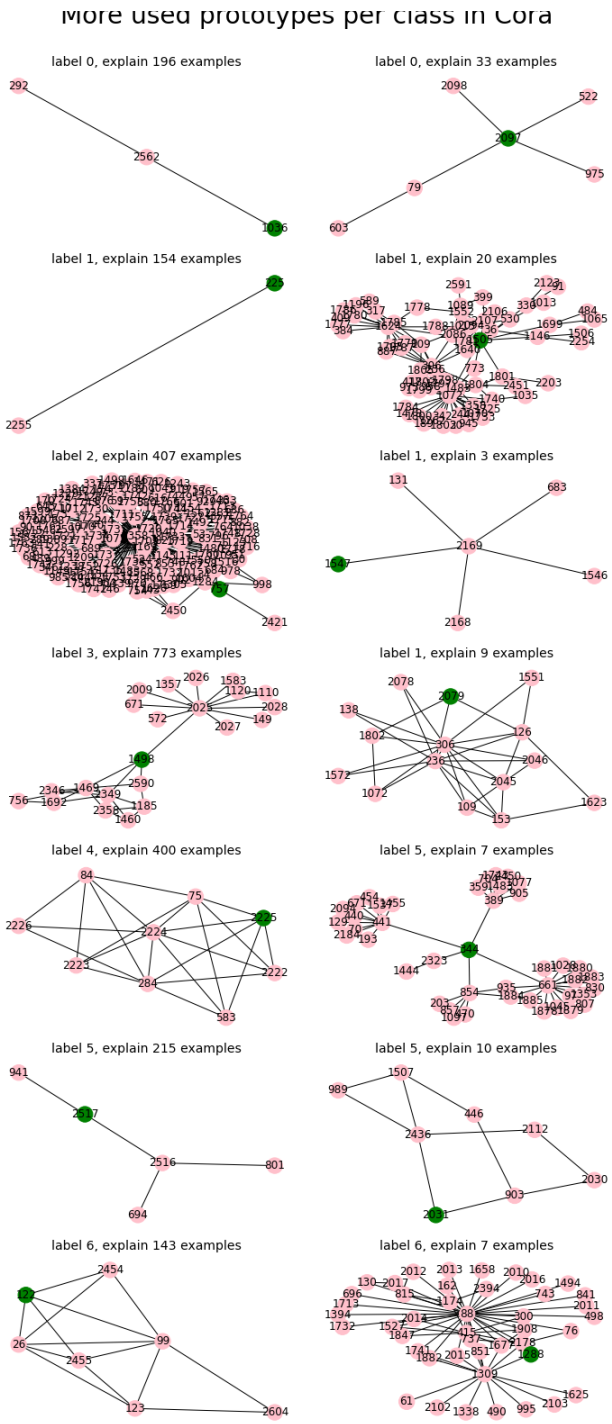

Figure 10. The two prototypes which are ranked first in similarity with samples of a class more times. Each row is a class. Each prototype shows its label and how many samples of the class represented by the row have it as the most similar prototype. They are the prototypes found by the sequential model trained on Cora.