Graph Neural Networks Go Forward-Forward

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present the Graph Forward-Forward (GFF) algorithm, an extension of the Forward-Forward procedure to graphs, able to handle features distributed over a graph's nodes. This allows training graph neural networks with forward passes only, without backpropagation. Our method is agnostic to the message-passing scheme, and provides a more biologically plausible learning scheme than backpropagation, while also carrying computational advantages. With GFF, graph neural networks are trained greedily layer by layer, using both positive and negative samples. We run experiments on 11 standard graph property prediction tasks, showing how GFF provides an effective alternative to backpropagation for training graph neural networks. This shows in particular that this procedure is remarkably efficient in spite of combining the per-layer training with the locality of the processing in a GNN.

1 Introduction

Backpropagation (BP, [3]) is the *de facto* standard algorithm for training neural networks and has been central to the success of deep learning. Despite its undeniable success, BP does have some drawbacks. For example, the BP algorithm uses the chain rule to compute gradients, which means that non-differentiable components cannot be included in neural networks, and its memory footprint is proportional to the total number of parameters in the model. Moreover, insufficient evidence to support that the BP algorithm could be a biological operation has motivated the investigation of *neuromorphic* methods for updating the parameters of neural networks [8, 20, 5]. The neuromorphic approach in deep learning is a growing trend that aims to relate deep learning methods to brain processes. The intent is to mimic the functioning of the human brain and the neurons that compose it. A major motivation behind the recent neurophormic development is the observation that the brain is highly capable and yet needs low energy compared to the computers needed to run and use large neural networks.

Recent work by [11] challenges the dominance of the BP algorithm by proposing the forward-forward (FF) algorithm as an alternative for training neural networks. The FF algorithm is inspired by Boltzmann machines [10] and by contrastive learning methods such as [9] and constitutes a better candidate in terms of biological plausibility. Initial experiments show that the FF algorithm achieves strong performance on vision datasets like MNIST and CIFAR-10, and that it can be extended to dealing with sequences [11].

Forward-forward algorithm. Instead of a forward and a backward pass, the FF algorithm performs a pair of forward passes to update the parameters of a network. The two passes run on two different datasets and have opposite objectives. The positive (negative) pass uses a positive (negative) dataset and adjusts, greedily layer by layer, the parameters of the neural network by maximizing (minimizing) a value, called the *goodness*. The FF algorithm is able to incorporate non-differentiable components between the layers and intends one day to enable the development of large power-efficient neural networks. The FF algorithm is able to handle both the supervised and unsupervised learning paradigms. This promising algorithm, however, remains largely unexplored at the moment.

Our main contribution is to improve and extend the FF algorithm to graph neural networks. We develop a recipe to calculate the *goodness* of any attributed graph and introduce the forward-forward algorithm on graph neural network (GFF). In the setting of graph property prediction, we show that GFF offers comparable performance to standard backpropagation on a range of different datasets from various domains. Additionally, we analyze different methods of computing the graph goodness, and of incorporating ground truth information in the message-passing framework. Finally, we provide statistics and discussions on the computational advantages of GFF. The code to reproduce our results will be made public upon publication.

2 Related Work

2.1 Graph Neural Networks

A graph G is a pair (V, E) where $V = \{v_1, ..., v_n\}$ is a set of nodes and E is a set of pairs of nodes (v_i, v_j) called edges. Graphs are ubiquitous in the real world and can be used to represent a large number of entities such as molecules, social networks, and maps. The abundance of problems related to graphs in numerous areas, including the natural and social sciences, has fuelled the development of methods to leverage the properties of graphs. Graph neural networks (GNNs, [17]) have quickly become the standard for dealing with graph-structured data. This large family of models has already generated several impressive achievements [6, 4, 23, 21]. GNNs operate through an iterative procedure called *message-passing* (MP). The representation of each node is updated, in parallel, by computing a function that aggregates the neighboring nodes. This procedure is repeated multiple times, where each iteration is parametrized by a layer in the GNN. The more layers there are, the more informed the remote nodes are of each other. After several message-passing iterations, a pooling function can be applied to aggregate the node feature vectors, with the resulting vector being used to make the final prediction.

2.2 The Forward-Forward algorithm

Positive and negative datasets. The FF algorithms requires encoding the label information in the data points. In [11], the one-hot encoded label is inserted in a corner of the image that contains no information. The correct label encoding is added to an image to form a positive sample. An incorrect label encoding is added to an image to form a negative sample. There is, of course, only one possible positive dataset. There can, however, be more than one negative dataset.

Goodness Function. One of the building blocks of the FF algorithm is the goodness function. This function is needed to update the parameters of the neural network. In [11], the goodness of a layer is the sum of the squared activities. The advantage of this definition is that it is simple and requires little computation. For example, the goodness function using squared activations is more efficient than the free energy function in the Boltzmann machines and has simple derivatives.

2.3 Contrastive Learning

The idea of contrastive learning is to learn by contrasting data that are similar and dissimilar [2]. This is often achieved by augmenting a dataset to create a similar, but slightly different, dataset. In computer vision, this augmentation can be done by transforming the images (cropping, rotating, and recoloring). This principle has been successfully adapted to GNNs in instances such as GraphCL [22] and MolCLR [19]. These methods require finding a method to compare two representations (vectors). In the FF algorithm, contrastive learning is applied by using positive and negative datasets described above. The advantage of FF-based algorithms is that it is not required to compare the (vector) representations, but only the (scalar) goodness values.

3 Method

This section describes the GFF recipe in detail. Our work focuses on the task of graph property prediction. The initial datasets are composed of multiple graphs that each belong to a label.

In the standard FF algorithm, the different labels are represented by a one-hot encoding scheme. In [11] and subsequent works, the label encodings are inserted as replacements for the first few pixels (which contain no information, of course) of the input images. In order to include label information in a graph, we develop and test two approaches. One consists in appending the one-hot encoded label to each node representation in the input graphs. This encoding scheme is displayed in Appendix B (Figure 4). A second approach, which is more graph-specific and which requires no data redundancy, is to endow input graphs with an additional *virtual node* carrying all information about the label. A virtual node is a node added to the graph and connected in some ways to the rest of the nodes. This encoding scheme is described in Figure 3. Note that the virtual node is connected bidirectionally to every other node in the graph. In order to allow the network to treat the virtual node as a separate entity, we use separate weight matrices for the message passing along the virtual node edges.

In order to generate negative samples for training with GFF, we simply encode the input graph with an incorrect label. Since the only difference between positive and negative data is the label, GFF should learn to ignore all features of the graph that do not correlate with the label.

Notation. Let n be the number of nodes in the input graph and suppose that this number remains the same throughout the layers of the network. Let d be the dimension of the activation vectors. Let $\mathbf{A}^{(i)} = \left(\mathbf{a}_1^{(i)}, \ldots, \mathbf{a}_n^{(i)}\right)$ be the activation matrix that contains the activation vectors $\mathbf{a}_j^{(i)} = \left(a_{j1}^{(i)}, \ldots, a_{jd}^{(i)}\right)$ of all the nodes of the i-th layer.

3.1 Goodness of a Graph

Learning on graphs using FF requires defining the goodness function for an attributed graph. This is the function that allows computing the layer-wise loss. Intuitively, the network is trained so that the goodness for positive samples is high (or higher than a threshold), while the goodness for negative samples is minimized. Moreover, the goodness should be computed in such a way that all information about it is discarded before passing the activations to the next layer. More information about how the goodness is used for training and inference is provided in Sections 3.3 and 3.4. One of the main differences with the approach proposed in [11] is that, when dealing with graphs, we carry a separate feature vector for each node, and not a single vector representing the whole sample. Since the goodness comes in the form of a single scalar for the whole graph, we need to find a way to aggregate these vectors. The aggregation function can be any permutation-invariant function of the set of node representations. Then, the mean of the elements of the resulting vector represents the goodness.

The goodness $g^{(i)}$ of the i-th layer, when we use sum-pooling as an aggregation function, is defined such that

$$g^{(i)} = \frac{1}{d} \sum_{k=1}^{d} \sum_{j=1}^{n} \left(a_{jk}^{(i)}\right)^2 . \tag{1}$$

Notice that more than one goodness function is possible, but the one above shines by virtue of being uncomplicated, and we find that it works well in practice. To summarize, the computation of the goodness is based on the following recipe:

1. Normalized activity vectors of all the nodes and the adjacency matrix are fed to the message-passing algorithm. This produces updated activity vectors. A nonlinear activation function is applied.
2. The activity vectors are squared.
3. The squared activity vectors are aggregated with a pooling function. This produces an aggregated vector.
4. The mean of the aggregated activity vector is computed and represents the goodness of the graph.

3.2 Architecture

Our framework can potentially be applied to any GNN. The layers can be convolutional or even attentional. In our experiments, the layers employed are convolutional layers found in the popular

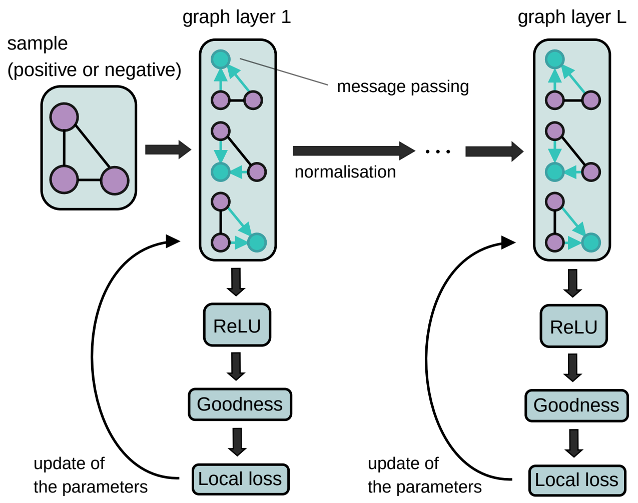

Figure 1: The architecture of an FFGNN. The layers are updated in a greedy manner. Each layer performs the MP procedure on each node (illustrated by the multiple graphs in each layer).

graph convolutional networks (GCN, [13]). The main feat of GCNs is to extend the convolution operator to non-Euclidean data in the form of graphs. In GCNs, for every node in the input graph, neighbor representations are aggregated with a position-invariant function. A linear transformation is applied to the resulting, aggregated vector, and the nodes are then updated with the new contextual representation. Specifically, the node-level update is described by the following equation:

$$\mathbf{x}_i' = \mathbf{\Theta}^\top \sum_{j \in \mathcal{N}(v) \cup \{i\}} \frac{\mathbf{x}_j}{c_{i,j}}, \tag{2}$$

where $\mathbf{x}_j$ is the vector representation of node i, $\mathcal{N}(i)$ is the set of neighbors of node i, $\mathbf{\Theta}$ is the weight matrix for the layer, $c_{i,j}$ is a normalization constant for the edges between node i and j, and $\mathbf{x}_i'$ is the updated node representation.

In the GFF algorithm, the activity vectors are normalized before being passed to the next layer. This is performed in order to avoid the information used to compute the goodness in the previous layer influencing the goodness in the next one. In this paper, the norm used is the length of the vector. Remember that $\mathbf{a}_j^{(i)} = \left(a_{j1}^{(i)}, a_{j2}^{(i)}, \ldots, a_{jd}^{(i)}\right)$ is the activity vector of the j-th node of the i-th layer. The norm is defined such that

$$\left\|\mathbf{a}_j^{(i)}\right\| = \sqrt{\left(a_{j1}^{(i)}\right)^2 + \left(a_{j2}^{(i)}\right)^2 + \cdots + \left(a_{jd}^{(i)}\right)^2}. \tag{3}$$

This norm has the advantage of being simple, but one could of course try other types of layer normalization.

3.3 Loss and Training

One of the main differences between a GFF-based GNN and a standard GNN is that the parameters of the former are updated layer by layer, instead of after going through all the layers. In other words, only one layer is being trained at each time. In this sense, the GFF algorithm can be said to be greedy or local layer-wise. Note that there is no predictive loss function. The objective is to maintain the goodness above a threshold if the data is positive and below this threshold if the data is negative. The idea is to minimize or maximize (depending if positive or negative) the local loss function of the layer. Let T be a threshold. For positive samples, the local loss function of the i-th layer is defined such that

$$L_i = \log\left(1 + \exp\left(-g^{(i)} + T\right)\right). \tag{4}$$

For negative samples,

$$L_i = \log\left(1 + \exp\left(g^{(i)} - T\right)\right). \tag{5}$$

The update can then be directed by the derivative of the local loss function. Note that this requires computing derivatives, but not applying the chain rule through all the layers. This is the reason that non-differentiable components can be added between the layers. This could allow, for example, symbolic components [1, 14]. Once the parameters of a layer are totally updated, it is the turn of the next layer to follow the same procedure. The activity vectors of the previous layer are passed to the next layer. The entire architecture can be visualized in Figure 1.

This greedy approach using a local loss function should result in the GFF algorithm being more memory efficient during training since we need to compute much fewer gradients and we don't need to keep the activations in memory for the backward pass. We elaborate more on this in Section 5. The optimization problem also gets simpler, given that each layer is solving a smaller task in a lower-dimensional parameter space.

3.4 Inference

The prediction phase in GFF is done in a quite different manner than in backprop-GNNs since there is no predictive loss function. Let us suppose that the goal is to classify some graph G. Let C be the number of possible labels. The idea is to first create the *label graph* G_{c_i} for each label c_i. The label graph G_{c_i} is created by appending the label encoding of the label c_i to all the nodes of the graph G. Let D_C be the augmented dataset containing the C label graphs. The next step is to compute the *label goodness* of each graph in the set D_C. The label goodness g_{c_i} of the label graph G_{c_i} is the sum of the goodness of all the layers of the FGNN that received G_{c_i} in input. Let L be the number of layers of the FGNN.

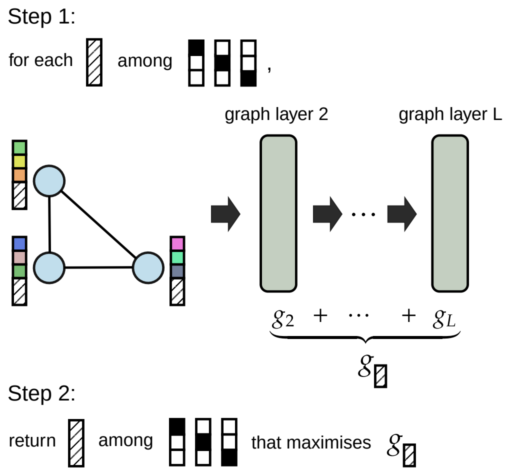

Figure 2: The prediction is done in two steps. Step 1: the label goodness is computed for each label. Step 2: the label selected is the label that has the highest label goodness.

More formally, the label goodness g_{c_i} is defined such that

$$g_{c_i} = \sum_{i=2}^{L} g^{(i)}. \tag{6}$$

Note that, similar to [11], the goodness of the first layer is not used in the prediction. This leads to better empirical results. The prediction is then accomplished by computing the label goodness of each of the label graph in the set D_C and then choosing the label that possesses the highest label goodness. More formally, the returned label $c^\star$ is selected such that

$$c^\star = \arg\max_{c_i} g_{c_i}. \tag{7}$$

For an illustration of the prediction scheme, see Figure 2. This type of prediction is interesting since the goodness of the other labels can also provide information. However, from a computational point of view, it can become difficult if the number of labels is very large. The adaptation of this kind of prediction scheme to a regression setting could be of value and is the subject of future work.

4 Experiments

We run multiple experiments on 11 datasets for graph property prediction. We compare the performances of GNNs trained with backpropagation with GNNs trained using GFF. We then perform several ablations to better understand and evaluate the building blocks of our method such as the pooling function, the goodness computation, and the label encoding. For datasets where the graph doesn't have node features, we assign the one-hot encoded node degree as the initial feature for each node.

All experiments are implemented in Pytorch/Pytorch Geometric [7, 16] on 1 NVIDIA 3090 GPU.

4.1 Datasets

The datasets used in this paper are mainly obtained from the TUDatasets collection [15]. The datasets range from relatively small ones such as **Peking_1** to larger ones such as **Yeast** and cover molecules (**BZR**, **COX2**, **MUTAG**, **SN12C**, **SW-620H**, **Yeast**), bioinformatics (**PROTEINS**, **Peking_1**), computer vision (**MSRC_9**) and social networks (**COLLAB**, **IMDB-Binary**). For more information on the datasets, see Table 6 in Appendix A.

4.2 Architecture and Training Details

In our experiments, all models contain 3 GCN layers with 128 hidden units per layer. We train the models for 200 epochs with a batch size of 128, using the Adam optimizer [12] with learning rate 10^{-3}. We report the mean and the standard deviation over 5 different random seeds.

4.3 Results

Dataset	**GCN**	**FF-GCN**
PROTEINS	0.60 ± 0.02	0.62 ± 0.10
IMDB-Binary	0.70 ± 0.05	0.63 ± 0.08
BZR	0.92 ± 0.03	0.89 ± 0.05
COX2	0.82 ± 0.04	0.78 ± 0.08
MUTAG	0.71 ± 0.04	0.71 ± 0.04
SN12C	0.96 ± 0.00	0.95 ± 0.00
SW-620H	0.95 ± 0.00	0.94 ± 0.00
Yeast	0.88 ± 0.00	0.88 ± 0.00
Peking_1	0.67 ± 0.11	0.59 ± 0.06
COLLAB	0.79 ± 0.01	0.68 ± 0.02
MSRC_9	0.91 ± 0.04	0.82 ± 0.09

Table 1: Comparison of the GCN (BP) and the FF-GCN (GFF). The results are in terms of test accuracy and have been obtained on 5 random seeds.

Dataset	**Speedup**
PROTEINS	22
IMDB-Binary	18
BZR	35
COX2	22
MUTAG	61
SN12C	2.5
SW-620H	2.7
Yeast	2.6
Peking_1	42
COLLAB	2.9
MSRC_9	28

Table 2: Comparison of training speed between backpropation and FF. The speedup value represents the factor by which the GFF algorithm is faster to train than the GCN for the same number of epochs.

Table 1 shows the comparison between GCNs trained with backpropagation and a GCNs adapted and trained with GFF. We find that the two networks perform very similarly in almost all cases. In one instance, we find that GFF outperforms the backpropagation alternative. Not that in this table for GFF we only show the use of sum-pooling to compute the goodness (see Section 3). However, for some datasets, we get even better results with mean pooling, which we show in the ablations (Section 4.4).

In Table 2, we report the difference in training speed between training with backpropagation and with FF. Note that we only compare the amount of time it takes to perform the same number of epochs on the datasets, and not the time it takes for the models to converge. For small datasets we observe speedups of more than an order of magnitude. For larger datasets, the time spent moving data in and out of the memory takes a significant portion of the training time, and the relative speedup of FF

Dataset	Concat	Virtual
PROTEINS	0.62 ± 0.10	0.61 ± 0.05
COLLAB	0.68 ± 0.02	0.52 ± 0.01

Table 3: Results for ablation on the label encoding method. Test accuracy is shown. The concatenation method outperforms the virtual node approach.

compared to backpropagation decreases. Nonetheless, on all datasets, FF requires at most half as long to perform the same number of training steps as backpropgation.

4.4 Ablations

Virtual node. As explained in Section 3, we test another method for encoding the label in the samples, based on virtual nodes. Each input graph is endowed with an virtual node that is connected bidirectionally to every other node. The virtual node features are initialized with a fixed embedding of the ground truth label for the graph. Additionally, we need the network to treat the virtual node as a separate entity, and thus learn to propagate its message separately. To this purpose, we use a Relational GCN layer (RGCN, [18]), which has separate weights for different edge categories. In our case, the virtual node edges are assigned different categories than standard edges. Finally, the goodness is computed by only considering the activities of the virtual node after message-passing. Conceptually, the virtual node approach has the advantage that it avoids duplicating the data, while also being more interpretable. Table 3 shows the results of the ablation on the **PROTEINS** and **COLLAB** datasets. We find that the node feature concatenation method still works better and more consistently. Additionally, we experimentally find that pairing an RGCN with our GFF loss causes the training to be quite unstable and prone to reaching infinite values in the exponential.

Computation of the Goodness. One of the key components of training with the FF algorithm is the notion of goodness of a (sample, label) pair. At each graph layer, there are multiple reasonable ways of computing the goodness. One can first square the representation component-wise and then sum over both the node and feature dimensions (Eq. 1). Alternatively, one can first sum over the node dimension to build one vector for the whole graph, square it component-wise, and finally sum it,

$$\hat{g}^{(i)} = \frac{1}{d} \sum_{k=1}^{d} \left(\sum_{j=1}^{n} a_{jk}^{(i)} \right)^2 . \tag{8}$$

In our experiments we computed the goodness values using Eq. 1. Table 4 shows how the choice the goodness computation influences performance on MSRC_9, IMDB-BINARY, BZR and COX2.

Dataset	square-pool-mean	pool-square-mean
MSRC_9	0.90 ± 0.05	0.80 ± 0.08
IMDB-BINARY	0.59 ± 0.06	0.50 ± 0.06
BZR	0.78 ± 0.02	0.77 ± 0.10
COX2	0.81 ± 0.05	0.77 ± 0.09

Table 4: Comparison of the different ways of computing the goodness. Eq. 1 corresponds to the column "square-pool-mean" and Eq. 8 correspond to the column "pool-square-mean".

Dataset	additive	mean
MSRC_9	0.90 ± 0.05	0.87 ± 0.07
IMDB-BINARY	0.59 ± 0.06	0.61 ± 0.07
BZR	0.78 ± 0.02	0.84 ± 0.07
COX2	0.81 ± 0.05	0.77 ± 0.06

Table 5: Comparison of additive-pooling and mean-pooling when computing the goodness. Note that mean-pooling and additive pooling are not equivalent as the graphs in the dataset might have a different number of nodes.

Mean vs. Additive Pooling. As discussed in the previous paragraph, one way or another a pooling step over the nodes of the graph has to be applied when computing the goodness. In our experiments, we used additive pooling. In this section, we compare how changing the pooling to mean-pooling influences performance. We didn't observe a statistically significant difference between these two pooling methods, the numerical results are given in Table 5. Note that additional pooling operations that we didn't investigate (for instance max-pooling) can also be used as alternative ways of computing the goodness.

5 Discussion

5.1 Memory Footprint

The memory footprint of training a graph neural network with GFF can be significantly lower that the backpropagation alternative. When using backpropagation, all of the activations have to be kept in memory to allow the propagation of the gradients during the backward pass. This results in memory consumption increasing as we add more layers to the network. For very large networks, this can occupy a very large amount of memory, making it impossible to train the model on a single GPU and requiring complex engineering to train in a distributed way. When training networks with FF, only a single layer is trained at a time, and thus only a subset of the activations need to be kept in memory. For deep networks, the difference in memory requirements can be significant. Provided that GFF scales, it would be possible to train very deep networks on very large graphs, such as social networks or protein complexes, with a fraction of the memory needed by backpropagation.

5.2 GFF and Top-down Effects

One weakness of GFF with respect to backpropagation, as outlined in [11], lies in the lack of top-down information flow. In practice, what is learned in later layers cannot affect what is learned in earlier layers. This is a limitation for GNNs trained with GFF, where long-range dependencies captured in later layers cannot inform the parameter updates of the initial layers. An initial recipe was proposed in [11]. Nonetheless, the graph structure of our data might allow augmenting the input with additional connections, that still allow capturing long-range context, despite missing top-down propagation. The investigation of this question is left for future work.

5.3 Generating Negative Data

Negative samples are fundamental to the functioning of the FF-based algorithms. As of now, the main method of coming up with negative samples has been to assign the wrong label to existing samples. However, this seems somehow limiting. Drawing from the contrastive learning literature, it might be possible to generate negative samples by modifying the underlying graph. This might for example help GFF focus on the longer-range correlations that have been shown to be vital for some GNN applications.

6 Conclusion

We introduced the Graph Forward-Forward algorithm, which extends the Forward-Forward algorithm to graph neural networks. The empirical findings in Section 4 demonstrate that our method is a viable alternative to backpropagation for GNNs. Despite some limitations, GFF carries several advantages as it's more biologically plausible and computationally more efficient in certain settings. Comprehensive analysis of the building blocks of our method further clarifies the inner workings of the models. This is quite encouraging and we hope and expect that it opens the doors for further research in this direction.

7 Future work.

In light of the novelty of the FF algorithm and the fact that this paper is the first one to extend the FF algorithm to graphs, there are, of course, many open problems and exciting possible directions to take in the future.

Goodness and local loss functions. A good question is how to choose the goodness function and the local loss function. In this paper, the squared activity vectors are added to each other, but it could be interesting to explore other aggregation methods. There are multiple options and it would not be

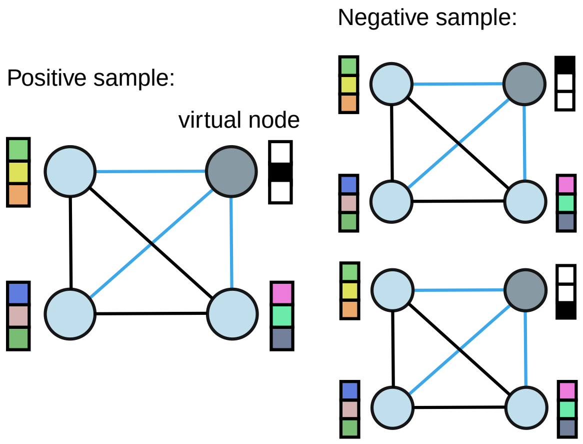

Figure 3: The same sample found in Figure 4, but transformed using a virtual node into a positive sample and two negative samples. On the left (positive sample): the correct label encoding has been added to the representation vectors of the nodes. On the right (negative sample): the incorrect label encoding has been added to the representation vectors of the nodes.

too surprising if it turned out that the choice of the goodness function and the local loss function depends on the specific application.

Extension to other graph tasks. This paper focuses on the supervised learning paradigm and more precisely on the graph property prediction task. The FF algorithm, and by extension the GFF algorithm, could also be used in an unsupervised setting. The unsupervised paradigm, however, raises some challenges in terms of how to create the negative datasets. In the supervised learning paradigm, nevertheless, the method could be extended to handle graph regression. The main issue here is how to define the negative dataset and how to perform the prediction, considering that the number of classes is no longer finite. Node and edge-level tasks are also a natural extension of the method. We leave it for future work.

References

[1] d'Avila Garcez Artur, R. Besold Tarek, de Raedt Luc, Földiak Peter, Hitzler Pascal, Icard Thomas, Kühnberger Kai-Uwe, C. Lamb Luis, Miikkulainen Risto, and L. Silver Daniel. Neural-symbolic learning and reasoning: Contributions and challenges. In *Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches: Papers from the 2015 AAAI Spring Symposium*, 2015.

[2] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E. Hinton. Big self-supervised models are strong semi-supervised learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[3] Ronald J. Williams David E. Rumelhart, Geoffrey E. Hinton. Learning representations by back-propagating errors. *Nature*, pages 533–536, 1986.

[4] Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis, and Pushmeet Kohli. Advancing mathematics by guiding human intuition with ai. *Nature*, 600:70–74, 12 2021.

[5] Giorgia Dellaferrera and Gabriel Kreiman. Error-driven input modulation: Solving the credit assignment problem without a backward pass. In *Proceedings of the 39th International Confer-*

ence on Machine Learning, volume 162 of *Proceedings of Machine Learning Research*, pages 4937–4955. PMLR, 17–23 Jul 2022.

[6] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, Peter W. Battaglia, Vishal Gupta, Ang Li, Zhongwen Xu, Alvaro Sanchez-Gonzalez, Yujia Li, and Petar Velickovic. ETA prediction with graph neural networks in google maps. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 3767–3776. ACM, 2021.

[7] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[8] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Spiking neural networks. *International Journal of Neural Systems*, 19(04):295–308, 2009. PMID: 19731402.

[9] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304. PMLR, 2010.

[10] G. E. Hinton and T. Sejnowski. Learning and relearning in boltzmann machines. In *Parallel distributed processing: Explorations in the microstructure of cognition*, pages 282–317–. MIT Press, Cambridge, MA, 1986.

[11] Geoffrey E. Hinton. The forward-forward algorithm: Some preliminary investigations. 2022.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[13] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[14] Luís C. Lamb, Artur S. d'Avila Garcez, Marco Gori, Marcelo O. R. Prates, Pedro H. C. Avelar, and Moshe Y. Vardi. Graph neural networks meet neural-symbolic computing: A survey and perspective. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4877–4884. ijcai.org, 2020.

[15] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *CoRR*, abs/2007.08663, 2020.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

[17] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[18] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks, 2017.

[19] Yuyang Wang, Jianren Wang, Zhonglin Cao, and Amir Barati Farimani. Molecular contrastive learning of representations via graph neural networks. *Nat. Mach. Intell.*, 4(3):279–287, 2022.

[20] James C.R. Whittington and Rafal Bogacz. Theories of error back-propagation in the brain. *Trends in Cognitive Sciences*, 23(3):235–250, 2019.

[21] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[22] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[23] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

A Dataset information

Figure 4 show summary statics and information about the datasets used.

Dataset	# of graphs	# of labels	avg # of nodes	avg # of edges	num feat. (node)	avg degree
PROTEINS	1113	2	39.06	72.82	4	7
IMDB-Binary	1000	2	19.77	96.53	501	8
BZR	405	2	35.75	38.36	53	4
COX2	467	2	41.22	43.45	53	4
MUTAG	188	2	17.93	19.79	7	4
SN12C	40004	2	26.08	28.11	65	–
SW-620H	40532	2	46.62	48.65	66	–
Yeast	79601	2	21.64	22.84	74	2
Peking_1	85	2	39.31	77.35	190	7
COLLAB	5000	3	74.49	2457.78	501	263
MSRC_9	221	8	40.58	97.94	10	9

Table 6: Information and summary statistics for the various datasets used in this paper.

B Graph representation and Label Encoding

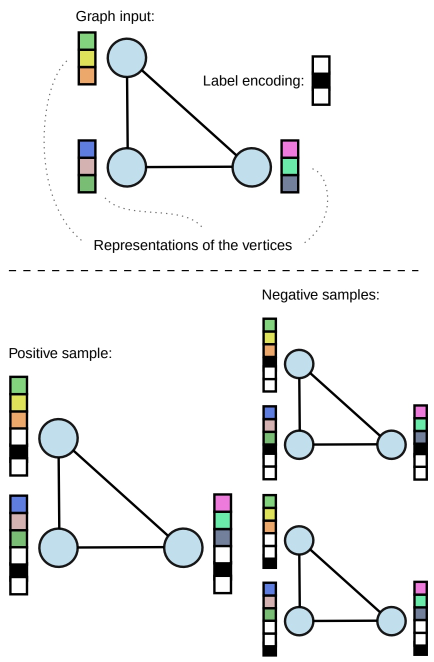

Figure 4: On the top: A graph input and the associated label encoding. The graph is composed of three nodes each having a representation vector. The same sample is transformed into a positive sample and two negative samples. On the bottom-left (positive sample): the correct label encoding has been added to the representation vectors of the nodes. On the bottom-right (negative sample): the incorrect label encoding has been added to the representation vectors of the nodes.