Over-Parameterized Neural Models based on Graph Random Features for fast and accurate graph classification

Nicolò Navarin[1][0000−0002−4108−1754], Luca Pasa[1][0000−0002−3023−3046], Claudio Gallicchio[2][0000−0002−6692−2564], Luca Oneto[3][0000−0002−8445−395X], and Alessandro Sperduti[1,4][0000−0002−8686−850X]

[1] University of Padua, Via Trieste 63, 35121, Padua, Italy
{nicolo.navarin,luca.pasa, alessandro.sperduti}@unipd.it
[2] University of Pisa, Largo Bruno Pontecorvo 3, 56127, Pisa, Italy
gallicch@di.unipi.it
[3] University of Genoa, Via Opera Pia 11a, 16145, Genoa - Italy
luca.oneto@unige.it
[4] University of Trento - DISI, Trento, Italy

Abstract. Recent works have proven the feasibility of fast and accurate time series classification methods based on randomized convolutional kernels [6, 38]. Concerning graph-structured data, the majority of randomized graph neural networks are based on the Echo State Network paradigm in which single layers or the whole network present some form of recurrence [9, 8]. This paper aims to explore a simple form of a randomized graph neural network. Our idea is implement a no-frills graph convolutional network and leave its weights untrained. Then, we aggregate the node representations with global pooling operators, obtaining an untrained graph-level representation. Since there is no training involved, computing such representation is extremely fast. We then apply a fast linear classifier to the obtained representations. We show that such a simple approach can obtain competitive predictive performance while being extremely efficient both at training and inference time. We also study when over-parameterization, namely generating more features than the ones necessary to interpolate, may be beneficial for the generalization abilities of the resulting models. Exploiting the algorithmic stability framework and based on empirical evidences from the considered graph datasets, we will shed some light on the over-parameterization setting.

Keywords: Graph Neural Network · Graph Convolution · Structured Data · Machine Learning on Graphs · Randomized Neural Networks.

1 Introduction

In this paper, we develop efficient graph neural networks for graph classification. When dealing with machine learning for structured data, there are typically two distinct families of tasks that can be tackled. The first task is node property prediction (e.g., *node* classification) which is defined as predicting one or more values associated with each node in a graph. Graph Neural Networks (GNNs)

have shown promising performance on such tasks [31]. The second task is graph property prediction (e.g., *graph* classification) in which the property that has to be predicted is a global property of a graph. In this case, the training set is composed of a set of graphs, each with the corresponding associated label. Such tasks require the inclusion of additional components in the GNN architecture, allowing the transformation of a set of node-wise representations to a single graph-level one before performing the prediction.

While many different architectures for node and graph classification have been proposed in the literature, the majority of the proposals have in common the end-to-end nature of their training, which results in fairly high computational complexity. Recently, in order to circumvent the need for expensive end-to-end training, a family of neural network models that are highly efficient to train have been receiving increasing attention. Most proposals along these lines so far have focused on the study of dynamic systems on discrete graphs in the area of Reservoir Computing (RC) [23, 26]. In this case, a reservoir layer randomly initialized under asymptotic stability constraints, and left untrained, is responsible for realizing the encoding of each node, while training is restricted only to the output *readout* layer [9, 8]. However, although this approach enables learning tasks on graphs in an extremely efficient way, it involves a fixed-point convergence process of the dynamic reservoir layer on the graph. This process, in turn, requires a number of iterations that can undermine the overall efficiency of the approach. More recently, it has been shown [32, 17] that the randomized approach on graphs can also be exploited without resorting to an iterative process. In particular, it is possible to perform node classification using randomized graph convolutions, i.e., without the necessity of training the convolution parameters, and still obtain competitive predictive performance when training is restricted to the readout part, as in RC-based approaches. While recently the approach has been proven feasible for tasks defined on graph nodes, it is still unclear if it also applies to graph classification tasks. This research question is not straightforward since the aggregation layer tends to lose a significant amount of information. A similar problem was already tackled in the case of randomized convolutions for time series analysis in a recent work [6]. In this paper, we show that by paying attention to some core aspects of the network architecture, it is possible to define a very efficient, randomized GNN model for graph classification tasks. Such critical components are: *(i)* the presence of non-linearity between graph convolution layers; *(ii)* the adoption of a *good* aggregation scheme; *(iii)* a wise initialization of the network weights and number of neurons. We analyze each one of these critical aspects in our ablation studies and show how each component influences the resulting predictive performance of the model. We refer to our proposed model as Untrained-GCN or U-GCN, acknowledging the intuitions behind it to come from the worlds of randomized neural networks and graph convolutional networks. The *readout* layer of the U-GCN consist of a linear classifier. This approach is commonly adopted when dealing with large-scale problems or aiming for computational efficiency. When dealing with large-scale problems or aiming for computational efficiency, a commonly adopted approach is to exploit feature sketching (random projections followed by a component-wise non-linearity) in conjunction with a linear classifier. The behavior of linear classifiers on increasing number of random features has been studied from the theoretical point of

view, in particular for ridge regression [4, 24] and based on stochastic gradient descent [22], showing, theoretically and empirically, the presence of the double descent and best-overfit phenomena [28, 37, 35, 12], namely the ability of these models to improve the generalization performance in over-parameterization, i.e., when having much more parameters than the ones needed to interpolate. To the best of authors' knowledge, no study in literature considers the case of random graph neural features. We speculate that the reason is that the majority of randomized graph networks in literature are based on a recurrent scheme of Reservoir Computing that, while generating expressive features, can result in a high computational complexity with hundreds or thousands of features [9]. The proposed U-GCN model can efficiently generate thousands of non-linear features. The classification (or regression) task is performed by a linear model, e.g. ridge classification, starting from the randomized features. This model allows us to study the behavior of untrained graph neural models when varying the number of generated features. In particular, we aim to understand when it may be convenient from the generalization performance point of view to generate a large number of random graph features (going beyond the interpolation threshold). For this purpose, we will leverage the Algorithmic Stability framework [28] and empirically show its potentiality in giving insights on the generalization ability of over-parameterized neural models based on graph random features.

1.1 Graph neural networks

A Graph Neural Network (GNN) [40, 14, 39] is a neural model that exploits the structure of the graph and the information embedded in feature vectors of each node to learn a representation $\mathbf{h}_v \in \mathbb{R}^m$ for each vertex $v \in V$. In many GNN models, the computation of $\mathbf{h}_v$ can be divided into two main steps: *aggregate* and *combine*. We can define aggregation and combination by using two functions, $\mathcal{A}$ and $\mathcal{C}$, respectively: $\mathbf{h}_v = \mathcal{C}(\mathcal{L}(v), \mathcal{A}(\{\mathcal{X}(u) : u \in \mathcal{N}(v)\}))$. The kind of aggregation function $\mathcal{A}$ and combination function $\mathcal{C}$ determinate the type of *Graph Convolution (GC)* adopted by the GNN. The first model that relies on graph convolutions was proposed by Micheli *et. al* in 2019 [25]. Recently, many novel GCs base model have been proposed [20, 5, 42, 15, 21, 44].

The model proposed in this paper is built on top of one of the most widely adopted GC operators, i.e. the GCN [20]: $\mathbf{H}^{(i)} = \mathcal{F}\left(\mathbf{S}\,\mathbf{H}^{(i-1)}\mathbf{W}^{(i)}\right), i > 1$ where $\mathbf{S} = \tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{I} + \mathbf{A})\tilde{\mathbf{D}}^{-\frac{1}{2}}$, $\mathbf{A}$ denotes the standard adjacency matrix of the graph G and $\tilde{\mathbf{D}}$ a diagonal degree matrix with the diagonal elements defined as $\tilde{d}_{ii} = 1 + \sum_j a_{ij}$. Further, $\mathbf{H}^{(i)} \in \mathbb{R}^{n \times m_i}$ is a matrix containing the representation $\mathbf{h}_v^{(i)}$ of all nodes in the graph (one per row) at layer i, $\mathbf{W}^{(i)} \in \mathbb{R}^{m_{i-1} \times m_i}$ denotes the matrix of the layer's parameters, and $\mathcal{F}$ is the element-wise (usually, nonlinear) activation function.

1.2 Graph neural networks with random weights

In structured data domains the models proposed in the last few years show increasing complexity, leading to novel architectures with a considerably high number of parameters. Unfortunately, this implies a high computational cost, especially in training the models.

For sequential data, many efficient architectures rely on the Reservoir Computing (RC) paradigm [23] which is based on exploiting fixed (randomized) values of the recurrent weights. The random weights are defined following the Echo State Property (ESP) [19] that ensures stability conditions of the dynamical system. In particular the Echo State Networks (ESN) [19], are widely used when an efficient recursive model is required. Gallicchio *et al.* in [8] proposed the first model for graph domain that exploits RC framework. The proposed model, dubbed GraphESN is composed of a non-linear reservoir and a feed-forward linear readout. The reservoir computes a fixed recurrent encoding function over the whole nodes of the graph as follows:

$$\mathbf{h}_v[t+1] = f(\mathbf{W}_{in}\mathbf{x}_v + \sum_{u \in \mathcal{N}(v)} \hat{\mathbf{W}}_h \mathbf{h}_u[t]), \tag{1}$$

where $\mathbf{W}_{in} \in \mathbb{R}^{m \times s}$, and $\hat{\mathbf{W}}_h \in \mathbb{R}^{m \times m}$. For each vertex $v \in V$, $\mathbf{h}_v[0]$ is initialized to $\mathbf{0} \in \mathbb{R}^m$. The computation of the global state $\mathbf{h}_v[t^*]$ involves the iteration of eq. (1) till $|\mathbf{h}_v[t^*+1] - \mathbf{h}_v[t^*]| \leq \epsilon$. Then, the global state is used by the readout of the model to compute the output using a linear projection: $\mathbf{o} = \mathbf{W}_{out} \sum_{v \in V} \mathbf{h}_v[t^*]$. In 2020 an evolution of the GraphESN is introduced in [9]. The FDGNN (Fast and Deep GNN) model constructs a progressively more abstract neural representation of the input graph by stacking successive layers of GNN. The formulation of this model is reminiscent of the original formulation of GNN [39] but the parameters of each layer are initialized by taking into account some stability constraints and then left untrained. Some simplifications of this kind of model were recently proposed [10].

In [32] the authors propose a model, dubbed Multi-resolution Reservoir Graph Neural Network (MRGNN) model, that exploits a Reservoir Convolutional layer for graphs able to simultaneously and directly consider all topological receptive fields up to $k - hops$. The convolutional layer relies on a multi-resolution[3, 33] structure that exploits nonlinear neurons followed by a standard feed-forward readout. The multi-resolution reservoir is defined as follows:

$$\mathbf{H}^r = \mathbf{H}^{k,\mathcal{T}}\mathbf{W}^r, \text{ where } \mathbf{H}^{k,\mathcal{T}} = [\underbrace{\mathbf{X}\mathbf{W}}_{\mathbf{H}^{k,\mathcal{T}}_{(0)}}, \underbrace{\sigma(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W})}_{\mathbf{H}^{k,\mathcal{T}}_{(1)}}, \underbrace{\sigma(\tilde{\mathbf{A}}\sigma(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W})\mathbf{W}))}_{\mathbf{H}^{k,\mathcal{T}}_{(2)}}, \dots],$$

σ is the *tanh* activation function, $\tilde{\mathbf{A}}$ is is a generic transformation of the adjacency matrix that preserves its shape, $\mathbf{W}^r$ is a randomly projection matrix and $\mathbf{H}^{k,\mathcal{T}}_{(i)}$ represents the i-th column block of $\mathbf{H}^{k,\mathcal{T}}$. Note that each $\mathbf{H}^{k,\mathcal{T}}_{(i)}$ contains information only about random walks of length exactly equal to i.

Recently, Huang et al. [17] explored randomized graph convolutions for the task of node classification (differently from this paper in which we consider the more challenging setting of graph classification). The authors propose a single-layer architecture defined as $Z = \sigma(A^2XW)\beta$, where σ is the sigmoid function, $W \in \mathbb{R}^{d \times m}$ is the (random) wight matrix for m hidden neurons (that is left untrained), and β are the trained output weights. Notice that, contrarily to many graph neural networks, authors propose to adopt a single hidden layer with an increased receptive field instead of non-linearly stacking multiple layers with a smaller receptive field.

1.3 Untrained convolutions for time series

The design of deep randomized neural networks represents one of the emerging topics in deep learning (see, e.g., [11]). The fundamental idea behind these approaches is to replace as much as possible the optimization of the parameters of a deep learning model with their randomization [36]. This usually results in a neural architecture in which hidden layers are initialized randomly and left untrained, while training algorithms operate only on the output *readout* layer. It is interesting to note how this paradigm, on the one hand, allows for the design of extremely efficient baselines and, on the other hand, allows for highlighting and exploiting the architectural biases of neural information processing models. Another advantage of this approach is its marked suitability for implementations in neuromorphic hardware [41] and, in general, in hardware with low computational resources, e.g., for AI applications of a pervasive nature [1].

When dealing with temporal information, i.e., for sequence processing, the paradigm of choice in this context is represented by Reservoir Computing (RC) [23], and in particular Echo State Networks [19, 18]. Here, the crucial idea is to build an RNN whose internal connections are randomly initialized under asymptotic stability constraints. As an alternative to the RC recurrent approach, the idea of exploiting randomized convolutions has recently been explored for time series analysis in the ROCKET model [6]. This is a method based on randomized one-dimensional convolutions for efficient feature extraction on time series, performing consistently well on a diverse range of datasets. The core contributions of the ROCKET model were: *(i)* showing that the approach of exploiting randomized convolutional kernels instead of learning them with backpropagation is feasible; *(ii)* the adoption of a new non-differentiable readout function, that associated with the more commonly adopted global max pooling, that showed an improvement in the overall predictive performance.

2 Untrained GCN for graph classification

In this section, we present our model for efficient graph classification. We start in Section 2.1 detailing our graph convolution layers and how they are combined. Then, in Section 2.2 we describe the pooling operators we decided to adopt, and finally in Section 2.3 we describe our readout and possible alternatives.

2.1 Untrained GCN feature extraction

As previously discussed in Section 1.2, recent results in literature have shown that for the task of semi-supervised nod classification, graph neural networks with random weights are a feasible option. However, it is known in the literature that for the problem of node classification, even simple models perform well [30, 34, 29]. In this paper, we propose a randomized architecture that is inspired by fully trained graph neural networks, including the non-linearity scheme. In particular, we propose to instantiate multiple graph convolution layers (see section 1.1), each one followed by an element-wise non-linear activation function.

Following the literature on untrained neural networks we decided to exploit the hyperbolic tangent activation function. We considered the simple and

widespread GCN definition (see Section 1.1). The hidden node representation computed by the l-th layer is defined as:

$$\mathbf{H}^{(l)} = tanh(\mathbf{S}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}), \tag{2}$$

where $\mathbf{S}$ is the normalized Laplacian adopted by the GCN, $\mathbf{W}^{(l)}$ are the layer parameters and $\mathbf{H}^0 = \mathbf{X}$. Note that we omit the bias terms for the sake of simplicity. The final node representations are obtained concatenating the representation computed by each graph convolution layer, i.e. $H = [H^{(1)}, \ldots, H^{(L)}]$, where L is the number of layers of the network. While we leave as a future work the exploration of other activation functions, in our ablation studies we consider the network without activation functions between layers and show that the non-linearity has a significant effect on the overall performance of our method. Our approach is in contraposition to Huang et al. [17] that instead apply the non-linearity only after the message passing phase. Crucially, the weight values in $\mathbf{W}^{(l)}$ in eq. 2 are initialized randomly and left untrained. For the random initialization, we resort to the widely adopted Glorot uniform approach [13]. In particular, to control the stability of the expansion of the input information through the successive layers in the architecture, we introduce a *gain* hyperparameter θ to control the effective scaling of $\mathbf{W}^{(l)}$. In the resulting process, a weight matrix of shape $n \times m$ will have entries sampled from a uniform distribution $\mathcal{U}(-a, a)$ where $a = \theta\sqrt{\frac{6}{n+m}}$. In our ablation study, we show that considering this hyperparameter significantly improves the predictive performance of the overall network.

2.2 The global pooling layer

The untrained graph convolution layer presented in eq. 2 produces node representations that include information about each node's local connectivity. To perform graph-level tasks we shall obtain a single representation for the whole graph. Usually, neural architectures for graph classification achieve this using global pooling operators, e.g. global *maximum, minimum* or *average* pooling. Notice that in the standard end-to-end training fashion, the pooling operators are required to be differentiable. Instead, if no gradient has to pass through the pooling operator, we are free to choose also non-differentiable options. This is the case for the ROCKET model, presented in Section 1.3. The authors proposed a non-differentiable pooling mechanism that, in the context of randomized 1-D convolutions, was shown to consistently improve the predictive performance compared to other widespread pooling operators. This operator is referred to as *Percentage of Positive Values* (PPV) and is defined as: $PPV(\mathbf{z}) = \frac{1}{n}\sum_{i=0}^{n-1} I[z_i > 0]$, where $I[z_i > 0]$ is the indicator function which value is 1 if $z_i > 0$, 0 otherwise.

As suggested in the original paper, we used as global pooling both the *global max pooling* and *PPV*, concatenating the resulting representations. Note that this choice doubles the size of the global graph representation compared to the representations of the single nodes provided in output by the untrained graph convolution. We conducted ablation studies to show the impact of different aggregation functions on the overall performance of our proposed method.

2.3 Efficient readout

As mentioned before, our focus in this paper is the development of efficient and effective neural network models for graph classification. As discussed in Sections 2.1 and 2.2, the network that computes the graph-level representation does not need to be trained. This leaves the only trained parameters of the model to be those in the readout, i.e., the function mapping from the graph-level representation to the appropriate output for the task. As it is common in deep randomized approaches [11], the representation component of the neural architecture includes multiple layers, while the readout component is shallow. In the case of classification tasks, we exploit one of the fastest linear classifiers in the literature, the Ridge classifier. In the binary case, this classifier follows the simple idea of mapping the two possible classes in $\{-1, 1\}$, and then treats the problem as a regression task, solved with ridge regression.

While other classifier choices may lead to improved results, in this paper we test only this very efficient classifier, and leave the exploration of other more complex readouts as future work.

3 Graph Random Features and Algorithmic Stability

In this section, we aim to study if and under which conditions it is convenient to develop overparametrized graph models when using graph random features. We seek an answer in recent research exploiting measures from statistical learning theory, such as the Algorithmic Stability, and exploring their relationship with the observed empirical behaviour of the generalization error.

From a random graph representation, it is possible to compute an approximation of a specific notion of Algorithmic Stability, the Hypothesis Stability, that, together with the training error, are able to give insights on the generalization error and are fast to compute [28]. Let $\mathbf{h}_g$ be the hidden representation for a graph computed by the model presented before, and $\mathbf{H}$ be the matrix collecting the representations of all training graphs. We can consider this representation as the input of a linear model (the readout). It has been shown that the Hypothesis Stability $\mathcal{A}$ is proportional to the conditioning of the Gramian matrix $\mathbf{H}\mathbf{H}^\intercal$, i.e., $\mathcal{A} \propto \mathit{Cond}(\mathbf{H}\mathbf{H}^\intercal)$ where Cond is a function computing the condition number of a matrix with eigenvalues λ_i, i.e., $\lambda_{\max}/\lambda_{\min}$. Thus, we can study the relationship between the approximation of the Hypothesis Stability of such representation and the generalization capabilities of the models trained on such representations. In fact, the smaller the training error and the smaller the stability, the higher the generalization ability of the learned model should be [28].

4 Results and discussion

In this section, we present our experimental setting. A critical point when comparing different models is the possible dataset augmentation that is applied, and the considered validation strategy. We decided to use a common setting for the chemical domain, where the nodes are labelled with a one-hot encoding of their atom type. The only exception is ENZYMES, where it is common to use 18

additional available features, and we followed this convention. Moreover, in the literature, different validation strategies have been applied, making it difficult to perform a fair comparison between the various methods.

For the reported results we follow the validation strategy in [7]. We estimate the performance of the U-GCN model by performing 10-fold cross-validation and repeating the whole procedure 5 times to account for the random initialization. To select the best model, we used the average accuracy of 10-fold cross-validation on the validation sets, and we used the same set of selected hyper-parameters for each fold. We did not perform an extensive hyperparameter search on the network architecture since our goal is to design an untrained GCN model whose performance is relatively stable on hyperparameter choice. For this reason, for U-GCN, we fixed the number of layers to four. As for the number of neurons, we set the number of hidden neurons to $5,000$ per layer. In Section 4.3 we will explore how different choices of this hyperparameter influence the results. Since we use four layers, and concatenate two different readouts, the resulting graph representation is of size $40,000$. Notice however that since the weights are not trained, we just have to perform the forward phase which is extremely fast. We then train the ridge regression classifier that depends on a regularization hyper-parameter α that we choose in the set $\{10^{-4}, 10^{5}\}$. We also select the θ parameter for weight initialization in the set $\{0.01, 0.1, 1, 3, 5, 10, 30, 50\}$. Finally we present some empirical evidences regarding the ability of Algorithmic Stability to explain the good generalization abilities of over-parameterized U-GCN.

4.1 Datasets

We evaluated U-GCN on commonly adopted graph benchmarks. We considered four datasets modeling bioinformatic problems: PTC [16], NCI1 [43], PROTEINS, [2], and ENZYMES [2]. We also considered two social network datasets: IMDB-B and IMDB-M [45]. PTC, and NCI1 involve chemical compounds represented by their molecular graphs, where node labels encode the atom type, and bonds correspond to edges. The prediction task for PTC concerns the carcinogenicity of chemical compounds for male rats. In NCI1 the task represent anticancer screens for cell lung cancer. PROTEINS and ENZYMES involve graphs that represent proteins. Amino acids are represented by nodes and the edges connect amino acids that in the protein are less than 6Å apart. All the prediction tasks are binary classification tasks, except for the ENZYMES dataset, that represents a 6-class classification of chemical compounds. IMDB-B and IMDB-M are composed of graphs derived from actor/actress and genre information of different movies on IMDB. The target value for each movie represents its genre. IMDB-B models a binary classification task, while IMDB-M considers three different classes. Nodes in the social datasets have no associated label.

4.2 Experimental results

In Table 1 we report the results of our experimental comparison. We considered seven datasets to allow for a comparison with many existing methods in the literature. We performed a pairwise Wilcoxon signed-rank test between our proposed *U-GCN* method and the others. We chose this test because our focus is to propose an efficient and effective alternative and we want to show that our method

Model \ Dataset	**PTC**	**NCI1**	**PROTEINS**	**D&D**	**ENZYMES**	**IMDB-B**	**IMDB-M**
FGCNN[27]	58.8±1.8	81.5±0.4	74.6±0.8	77.5±0.9	-	-	-
DGCNN[27]	57.1±2.2	73.0±0.9	74.0±0.4	78.1±0.7	-	-	-
DGCNN[7] *	-	76.4±1.7	72.9±3.5	76.6±4.3	38.9±5.7	53.3±5.0	38.6±2.2
SGC[33] *	55.6±7.6	76.3±2.5	75.4±3.4	77.1±4.4	31.3±5.6	66.4±5.5	43.3±3.4
Cheb-Net[33]	55.2±6.5	80.9±1.9	75.8±5.1	77.9±3.7	38.1±6.2	70.6±3.8	43.9±3.4
GIN[7] *	-	80.0±1.4	73.3±4.0	75.3±2.9	59.6±4.5	66.8±3.9	42.2±4.6
DIFFPOOL[7] *	-	76.9±1.9	73.7±3.5	75.0±3.5	59.5±5.6	68.3±6.1	45.1±3.2
GraphSAGE[7]	-	76.0±1.8	73.0±4.5	72.9±2.0	58.2±6.0	69.9±4.6	47.2±3.6
Baseline[7]	-	69.8±2.2	75.8±3.7	**78.4±4.5**	65.2±6.4	50.7±2.4	36.1±3.0
FDGNN[9]	**63.4±5.4**	77.8±1.5	**76.8±2.9**	-	-	**72.4±3.6**	50.0±1.3
MGN[10]	-	78.8±2.3	-	-	-	72.7±3.2	49.5±2.2
GRN [10]	-	78.2±2.2	-	-	-	71.7±2.8	**50.5±1.4**
GESN[10]	-	77.8±2.0	-	-	-	71.7±3.6	48.7±2.1
MRGNN[32]	57.6±10.0	80.6±1.9	75.8±3.5	-	68.2±6.9	72.1±3.6	46.9±3.7
U-GCN	61.2±2.2 ($\theta = 0.1$)	**82.2±0.4** ($\theta = 30$)	74.2±1.4 ($\theta = 10$)	78.0±1.0 ($\theta = 5$)	**68.8±0.6** ($\theta = 3$)	68.7±1.2 ($\theta = 1$)	45.8±0.6 ($\theta = 1$)

Table 1: Experimental comparison between the proposed U-GCN and many state-of-the-art methods.

performs comparably to the state of the art. Thus, the absence of a statistically significant performance difference between our method and the alternatives is already a good result in our point of view. From the test, it emerges that our method performs even significantly better than some state-of-the-art end-to-end trained architectures, showing that the approach we propose is indeed promising.

In Table 2 we perform an ablation study to show the contribution of each core component of our architecture. First, we consider a version of U-GCN that only uses the global max pooling as an aggregator, thus discarding the PPV. For this ablation, we doubled the number of neurons in the network to consider graph representations of the same size. While there is no clear winner in the comparison, notice that the feature extraction of U-GCN is faster since it requires extracting half the number of features. The second ablation we consider is the same U-GCN where the *tanh* activation function between graph convolutional layers is removed, obtaining a linear model. In this case, U-GCN performs significantly better than linear ablation. Finally, we consider the impact of the θ parameter comparing U-GCN with a version where we fix $\theta = 1$ (its default value). In this case, U-GCN performs again significantly better than the ablation.

Concerning the computational times, running on CPU on a server equipped with an Intel(R) Xeon(R) CPU E5-2630L v3 @ 1.80GHz, for instance for the ENZYMES dataset with $5,000$ neurons per layer the feature extraction on the whole dataset takes 33 seconds, while a single LS-SVM training takes on average 5 seconds. For NCI1, the times are 42 and 6 seconds, respectively. These times are orders of magnitude faster when compared to GNN models trained end-to-end with stochastic gradient descent. Concerning the test times, they correspond to the forward pass and the evaluation of the Ridge regression model, thus they are roughly equivalent to the ones of common GNN models. Notice that the forward pass could also be implemented on GPU for even faster feature extraction.

Model \ Dataset	PTC	NCI1	PROTEINS	D&D	ENZYMES	IMDB-B	IMDB-M
U-GCN	61.2±2.2 ($\theta = 0.1$)	82.2±0.4 ($\theta = 30$)	74.2±1.4 ($\theta = 10$)	78.0±1.0 ($\theta = 5$)	68.8±0.6 ($\theta = 3$)	68.7±1.2 ($\theta = 1$)	45.8±0.6 ($\theta = 1$)
U-GCN ablation (max aggr.)	64.1 ± 1.5 ($\theta = 50$)	80.6 ± 0.5 ($\theta = 30$)	74.7 ± 0.9 ($\theta = 3$)	74.7 ± 0.8 ($\theta = 30$)	70.1 ± 0.8 ($\theta = 5$)	69.8 ± 1.1 ($\theta = 1$)	45.8 ± 0.7 ($\theta = 0.01$)
U-GCN ablation (linear) *	60.6 ± 1.0 ($\theta = 1$)	80.5 ± 0.3 ($\theta = 30$)	73.3 ± 0.7 ($\theta = 50$)	76.8 ± 0.5 ($\theta = 30$)	65.7 ± 1.9 ($\theta = 10$)	65.5 ± 1.9 ($\theta = 30$)	45.5 ± 1.38 ($\theta = 1$)
U-GCN ablation ($\theta = 1$) *	60.8 ± 1.3	80.2 ± 0.5	73.9 ± 0.5	77.2 ± 0.6	67.6 ± 1.3	68.7 ± 1.2	45.8 ± 0.62

Table 2: Ablation study: comparison of U-GCN with different variations.

4.3 Algorithmic Stability of over-parameterized U-GCN.

We study the behaviour of the model described in Section 2 varying the number of neurons (parameters) for certain configurations of the hyperparameters. Due to space constraints only a subset, the most informative, of the results are reported. We fixed the number of layers to four. We plot the performance varying the number of neurons for each layer from 10 to $10,000$ ($5,000$ for D&D) per layer. Since we use four layers and concatenate two different readouts, the resulting graph representation is up to size $80,000$ ($40,000$ for D&D). However, since the weights are not trained, we just have to perform the forward phase which is extremely fast even with a high number of features to extract. Then, we trained a ridge classifier characterized by a regularization hyperparameter α taking values in the set $\{0, 10^{-4}, \ldots, 10^{5}\}$. We also considered multiple values of θ, for weight initialization, in the set $\{0.01, 0.1, 1, 3, 5, 10, 30, 50\}$. Among the different graph classification benchmark datasets available we considered three datasets related to bio-informatics: ENZYMES [2], D&D [2], and NCI1 [43].

In this section, we report for different datasets and different hyperparameters configurations that can reach competitive performance, the training, validation, and test accuracies, varying the number of neurons. We also report the Algorithmic Stability estimated via the condition number of the Gram matrix (see Section 2), and the interpolation threshold (i.e., the value of the number of neurons such that the accuracy on the training set is 100% without regularization).

Figure 1 reports the results for the ENZIMES dataset for different values of θ. From Figure 1 we can see that there are two different regimes with a phase change. The first regime is the under-parameterized one, in which the actual dimension of the feature space is smaller than the interpolation threshold. In this setting there is a trade-off between accuracy, number of neurons and error, typical of the classical bias-variance trade-off [28]. Note that in this setting the Algorithmic Stability is, relatively, quite high. The second regime is the over-parameterized one, which is the one after the interpolation threshold, that is characterized by two new phenomena. The first one is that the accuracy on the test set starts to increase even if the model is interpolating (double-descent or best-overfit behavior [28]) but in correspondence of the interpolation threshold there is a change of phase in the Algorithmic Stability which suddenly drops around this threshold and then generally continues to decrease after the drop. In other words, Algorithmic Stability is able to tell us that adding more neurons can actually improve generalization instead of hurting it: in fact, in the over-parameterized regimes, accuracies increase while stability decreases which is a

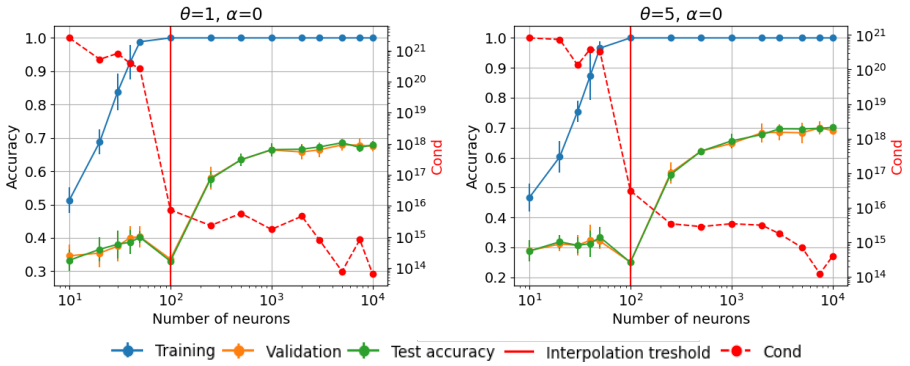

Fig. 1: ENZYMES dataset.

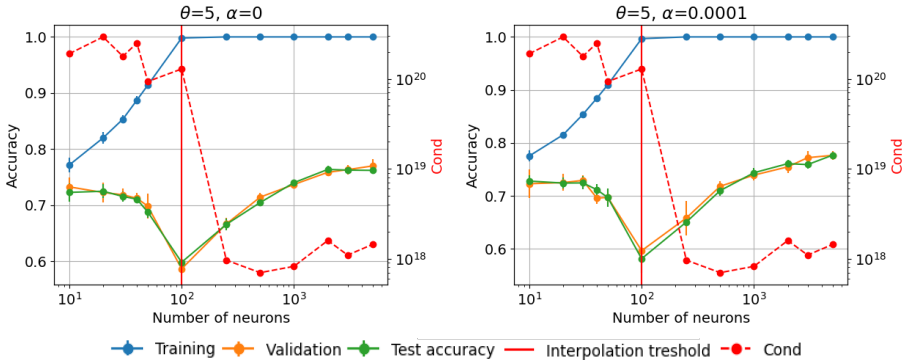

Fig. 2: D&D dataset.

clear sign of increasing generalization [28]. Figures 2 and 3 report the results of the D&D and NCI1 datasets. We can come up with similar observations as for the ENZYMES dataset, confirming the empirical evidence that the Algorithmic Stability is able to explain, and suggest, when over-parameterization can be beneficial for the generalization ability of neural models based on graph random features. Note also, that best performances are not always reached with simple empirical risk minimization and sometimes regularization ($\alpha > 0$) is needed but the Algorithmic Stability is always able to provide the necessary insights.

5 Conclusions

We proposed an extremely efficient GNN model for graph classification. The proposed architecture (U-GNN) is reminiscent of the models that rely on Reservoir Computing (RC). Indeed, as the name suggests, the U-GNN exploits simple stacked graph convolutional layers where the weights are randomly initialized and then left untrained. The random convolutional projections of the graph's nodes are aggregated to obtain a graph-level representation using a global pool-

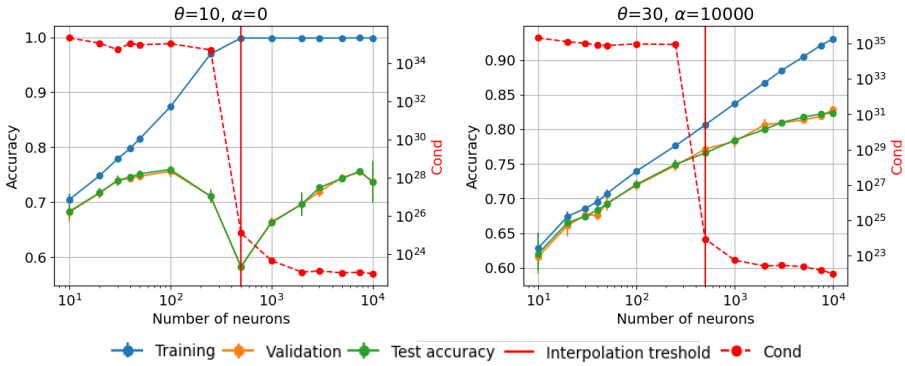

Fig. 3: NCI1 dataset.

ing operator that is the concatenation of the global max pooling and the percentage of positive values. The main advantage of the approach is that to compute the graph representations no training is required. Finally, the classification task is performed simply using ridge regression. We assessed the performance of U-GNN on 7 datasets from different application areas, comparing our proposal both with models that exploit standard end-to-end training and with GNN based on the RC framework. The empirical results show that our approach achieved results comparable to the state-of-the-art methods. We also investigated the generalization abilities of over-parameterized U-GNN, aiming to understand when over-parameterization, namely generating more features than the ones necessary to interpolate, may be beneficial for the generalization of the resulting models. For this purpose, we rely on the Algorithmic Stability framework that together with empirical evidences from several commonly adopted graph datasets helped us understand why more parameters can improve generalization. Of course, this work is a preliminary but promising step in understating over-parameterized neural models based on graph random features and more theoretical and empirical evidences need to be derived.

6 Acknowledgements

This work was partly funded by: the SID/BIRD project *Deep Graph Memory Networks*, Department of Mathematics, University of Padua; the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU; the project “iNEST: Interconnected Nord-Est Innovation Ecosystem" funded under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.5 - Call for tender No. 3277 of 30 December 2021 of Italian Ministry of University and Research funded by the European Union – NextGenerationEU, project code: ECS00000043, Concession Decree No. 1058 of June 23, 2016, CUP C43C22000340006; the project “EMERGE", funded by EU Horizon research and innovation programme (grant n. 101070918)"

References

1. Bacciu, D., Akarmazyan, S., Armengaud, E., Bacco, M., Bravos, G., Calandra, C., Carlini, E., Carta, A., Cassarà, P., Coppola, M., et al.: Teaching-trustworthy autonomous cyber-physical applications through human-centred intelligence. In: 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS). pp. 1–6. IEEE (2021)
2. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**(suppl_1), i47–i56 (2005)
3. Chen, L., Chen, Z., Bruna, J.: On graph neural networks versus graph-augmented mlps. In: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net (2021), https://openreview.net/forum?id=tiqI7w64JG2
4. Chen, Z., Schaeffer, H.: Conditioning of Random Feature Matrices: Double Descent and Generalization Error. arXiv preprint arXiv:2110.11477 (2021)
5. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: NIPS. pp. 3844–3852 (2016)
6. Dempster, A., Petitjean, F., Webb, G.I.: ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. Data Mining and Knowledge Discovery **34**(5), 1454–1495 (2020). https://doi.org/10.1007/s10618-020-00701-z
7. Errica, F., Podda, M., Bacciu, D., Micheli, A.: A fair comparison of graph neural networks for graph classification. In: International Conference on Learning Representations (2020)
8. Gallicchio, C., Micheli, A.: Graph Echo State Networks. Neural Networks (IJCNN), The 2010 International Joint Conference on pp. 1–8 (2010). https://doi.org/10.1109/IJCNN.2010.5596796
9. Gallicchio, C., Micheli, A.: Fast and deep graph neural networks. In: AAAI. pp. 3898–3905 (2020)
10. Gallicchio, C., Micheli, A.: Ring reservoir neural networks for graphs. arXiv preprint arXiv:2005.05294 (2020)
11. Gallicchio, C., Scardapane, S.: Deep randomized neural networks. In: Recent Trends in Learning From Data: Tutorials from the INNS Big Data and Deep Learning Conference (INNSBDDL2019). pp. 43–68. Springer (2020)
12. Ghorbani, B., Mei, S., Misiakiewicz, T., Montanari, A.: Linearized two-layers neural networks in high dimension. The Annals of Statistics **49**(2), 1029–1054 (2021)
13. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 9, pp. 249–256. PMLR, Chia Laguna Resort, Sardinia, Italy (13–15 May 2010), https://proceedings.mlr.press/v9/glorot10a.html
14. Gärtner, T.: A survey of kernels for structured data. ACM SIGKDD Explorations Newsletter **5**(1), 49 (Jul 2003). https://doi.org/10.1145/959242.959248, citation Key: Gartner2003 publisher-place: New York, NY, USA
15. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: NIPS. pp. 1024–1034 (2017)
16. Helma, C., King, R.D., Kramer, S., Srinivasan, A.: The predictive toxicology challenge 2000–2001. Bioinformatics **17**(1), 107–108 (2001)
17. Huang, C., Li, M., Cao, F., Fujita, H., Li, Z., Wu, X., Li, M.: Are Graph Convolutional Networks With Random Weights Feasible? IEEE Transactions on Pattern Analysis and Machine Intelligence **45**(3), 2751–2768 (2023). https://doi.org/10.1109/tpami.2022.3183143

18. Jaeger, H.: The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science (2001)
19. Jaeger, H., Haas, H.: Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. science **304**(5667), 78–80 (2004)
20. Kipf, T.N., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. In: ICLR. pp. 1–14 (2017). https://doi.org/10.1051/0004-6361/201527329
21. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated Graph Sequence Neural Networks. In: ICLR (2016). https://doi.org/10.1103/PhysRevLett.116.082003
22. Liu, F., Suykens, J., Cevher, V.: On the double descent of random features models trained with sgd. In: Neural Information Processing Systems (2022)
23. Lukoševičius, M., Jaeger, H.: Reservoir computing approaches to recurrent neural network training. Computer Science Review **3**(3), 127–149 (2009)
24. Mei, S., Montanari, A.: The generalization error of random features regression: Precise asymptotics and the double descent curve. Communications on Pure and Applied Mathematics **75**(4), 667–766 (2022)
25. Micheli, A.: Neural network for graphs: A contextual constructive approach. IEEE Transactions on Neural Networks **20**(3), 498–511 (2009)
26. Nakajima, K., Fischer, I.: Reservoir computing. Springer (2021)
27. Navarin, N., Tran, D.V., Sperduti, A.: Learning kernel-based embeddings in graph neural networks. In: European Conference on Artificial Intelligence (2020)
28. Oneto, L., Ridella, S., Anguita, D.: Do we really need a new theory to understand over-parameterization? Neurocomputing (2023)
29. Pasa, L., Navarin, N., Erb, W., Sperduti, A.: Empowering Simple Graph Convolutional Networks. IEEE Transactions on Neural Networks and Learning Systems **PP**(99), 1–15 (2023). https://doi.org/10.1109/tnnls.2022.3232291
30. Pasa, L., Navarin, N., Sperduti, A.: Simple Multi-resolution Gated GNN. 2021 IEEE Symposium Series on Computational Intelligence (SSCI) **00**, 01–07 (2021). https://doi.org/10.1109/ssci50451.2021.9660046
31. Pasa, L., Navarin, N., Sperduti, A.: Deep learning for graph-structured data. In: HANDBOOK ON COMPUTER LEARNING AND INTELLIGENCE: Volume 2: Deep Learning, Intelligent Control and Evolutionary Computation, pp. 585–617. World Scientific (2022)
32. Pasa, L., Navarin, N., Sperduti, A.: Multiresolution reservoir graph neural network. IEEE Trans. Neural Networks Learn. Syst. **33**(6), 2642–2653 (2022). https://doi.org/10.1109/TNNLS.2021.3090503, https://doi.org/10.1109/TNNLS.2021.3090503
33. Pasa, L., Navarin, N., Sperduti, A.: Polynomial-based graph convolutional neural networks for graph classification. Mach. Learn. **111**(4), 1205–1237 (2022). https://doi.org/10.1007/s10994-021-06098-0
34. Pasa, L., Navarin, N., Sperduti, A.: Compact graph neural network models for node classification. Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing pp. 592–599 (2022). https://doi.org/10.1145/3477314.3507100
35. Poggio, T., Kur, G., Banburski, A.: Double descent in the condition number. arXiv preprint arXiv:1912.06190 (2019)
36. Rahimi, A., Recht, B.: Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. Advances in neural information processing systems **21** (2008)
37. Rangamani, A., Rosasco, L., Poggio, T.: For interpolating kernel machines, minimizing the norm of the erm solution minimizes stability. arXiv preprint arXiv:2006.15522 (2020)

38. Rodrigues, I.R., Neto, S.R.d.S., Kelner, J., Sadok, D., Endo, P.T.: Convolutional Extreme Learning Machines: A Systematic Review. Informatics **8**(2), 33 (2021). https://doi.org/10.3390/informatics8020033
39. Scarselli, F., Gori, M., Ah Chung Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The Graph Neural Network Model. IEEE Transactions on Neural Networks **20**(1), 61–80 (2009). https://doi.org/10.1109/TNN.2008.2005605
40. Sperduti, A., Starita, A.: Supervised neural networks for the classification of structures. IEEE Trans. Neural Networks **8**(3), 714–735 (1997). https://doi.org/10.1109/72.572108
41. Tanaka, G., Yamane, T., Héroux, J.B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., Hirose, A.: Recent advances in physical reservoir computing: A review. Neural Networks **115**, 100–123 (2019)
42. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
43. Wale, N., Watson, I.A., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. Knowledge and Information Systems **14**(3), 347–375 (2008)
44. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How Powerful are Graph Neural Networks? In: International Conference on Learning Representations (2019)
45. Yanardag, P., Vishwanathan, S.: Deep graph kernels. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15 pp. 1365–1374 (2015). https://doi.org/10.1145/2783258.2783417