Final Report - Graph Neural Networks

Pedro Ferreira
Instituto Superior Técnico
Lisbon, Portugal
pedroalexxxferreira@gmail.com

Pedro Silva
Instituto Superior Técnico
Lisbon, Portugal
pedrotomaz20@gmail.com

Abstract

We provide an historical overview of the works on Graph Neural Networks (GCNs) and we mainly focus on the work in Kipf and Welling (2017). We implement it in Pytorch and reproduce some results within a semi-supervised setting using the Karate Club dataset from Zachary (1977). After demonstrating the potential of the GCN, we experiment with improving the performance by adding an attention mechanism and compare it with the GCN. In the end we implement a suggested improvement in Kipf and Welling (2017) and conclude with a replication of results in Kipf and Welling (2017) and some final remarks on the work we produced.

1 Introduction

The domains of images, sound and sentences have served as a basis structure in Machine Learning research for decades. The graph data structure however, as is usually visualized, has been relatively unexplored and neglected due to the massive use cases of the usual domains. However, graphs are everywhere. They are so ubiquitous that the previous data structures are particular cases of graphs. An image is a pixel lattice, a specific type of graph that has some properties that allows us to disregard its general structure and capitalize on what is immediately visible. A sound is a sequence of numbers representing amplitudes or frequencies indexed in time, a specific type of directed graph, a graph where the edges between vertices are one-directional. The same goes for a sentence, a time indexed sequence of symbols which can be represented as one-hot vectors. To formalize this point of view, we first need to understand what are graphs and the mathematics behind them.

1.1 Graphs

A graph is a set of objects with some degree of relatedness among them. It is usually mathematically represented as an ordered pair $G = (V, E)$ where V is the set of vertices (also called nodes) and E is the set of edges that represent the degree of relatedness between the vertices i.e. are associations of two vertices. One of the most useful representations of graphs is by the way of an adjacency matrix A which is $|V| \times |V|$ elements. In an unweighted graph (one where the existing edges are all the same), the adjacency matrix has one in position (i, j) if there exists an edge between vertices i and j and has a zero otherwise. Weighted graphs allow for all real numbers in the elements of the adjacency matrix. A degree matrix D is a $|V| \times |V|$ diagonal matrix with diagonal entries defined as the number of edges that terminate at the vertex corresponding to that entry. A graph Laplacian $\mathcal{L}$ is defined as $\mathcal{L} := D - A$. The symmetric Laplacian $\mathcal{L}^{sym}$ is defined as $\mathcal{L}^{sym} := D^{-\frac{1}{2}}\mathcal{L}D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$. The use of these quantities is widespread in algorithm construction and will be used further in this paper.

As abstract as this idea sounds, most data structures found in society and nature are actually of this form. From social networks, gene regulatory networks, molecules, Feynman diagrams in particle physics and the before seen images and sentences are all graph-like. Thus, it is of extreme importance that models exist that can be fed this data structure.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

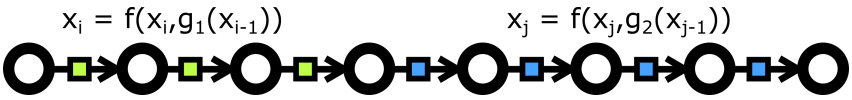

Figure 1: The messages are recurrently combined until a final output containing the messages of every node is created. At every time step, for every vertex, the message coming from the left neighbour is processed by the edge type and combined with the message currently in that vertex to produce the next message at that vertex. The processing is dependent upon the type of edge, hence the different g_i functions and colors on the edges. The combination is the f function operating on the processed neighbour message and the current message at that vertex.

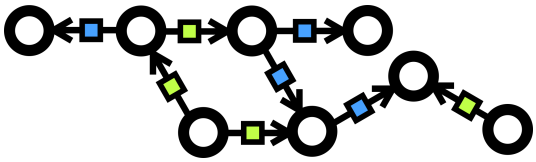

Figure 2: Messages in this setting are transmitted through the directed edges, processed by those same edges and combined at the incoming edge. Thus, information travels through the graph similar to heat diffusion.

Several models have been proposed over the years and we will now briefly take a look at those. Something that may seem curious is that one can distinguish two different ways to create models that act on graphs: a recurrent and convolutional perspectives. Although curious, it should not be surprising, since it is what research has been about in the past and with great success so far.

1.2 The Recurrent Perspective

The first perspective is based on the generalization of recurrent neural networks (RNNs) to work on graph data in Scarselli et al. (2009). We can visualize this RNN perspective as a directed chain graph (Figure 1), where the vertices are recurrent units. A message $\boldsymbol{x}_i$, a vector representation of the data at that point, is passed from the left to right using the relation $\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{g}(\boldsymbol{x}_{i-1}))$. We can let the edges between the recurrent units vary, depending on the type of edge being considered. That is to say, $\boldsymbol{g}$ can vary from edge to edge, as we would expect if we were treating the problem of molecules and the different atomic bonds (simple, double, ...). To generalize this to graphs, we simply have to allow the transmission of messages to neighbouring nodes depending on the adjacency matrix of the graph. In Figure 2 we can easily see how this generalization can be made. Messages travel through the graph in a similar manner as the chain graph described before and information is thus transformed as a function of the graph structure.
The initial idea proposed in Scarselli et al. (2009) was to do as many time steps as were required to reach a fixed point. One problem with this is that one would have to design the neural networks with some restrictions so that convergence is guaranteed. Another problem is that propagation of information of graphs is exponential with the distance between nodes, meaning there was room for improving by the use of some sort of approximation. In Li et al. (2016) they used Gated Recurrent Units (GRUs) and Long Short Term Memories (LSTMs) architectures on top of the previous work and achieved state-of-the-art performance on a problem from program verification, in which subgraphs need to be described as abstract data structures.

1.3 The Convolutional Perspective

Another way of approaching the problem, which was done in parallel to the previous approach, was to apply what we know of convolutional neural networks (CNNs) and generalize it to work with graphs. The work of Bruna et al. (2013) shows two ways to construct deep neural networks to work on graphs. One of these, what they called *spectral construction*, describes how a convolution operator can be defined on the Fourier transform of the graph. In $\mathbb{R}^d$, a convolution is a linear operator diagonalized by the Fourier basis $e^{i\omega \cdot t}, \omega, t \in \mathbb{R}^d$. The work in Belkin and Niyogi (2001) extended this idea to graphs. They found the graph Laplacian, an operator which provides an harmonic analysis of the graph itself, was the way to get this Fourier basis. Since $\mathcal{L}$ is a real symmetric positive semidefinite matrix, it

has a complete set of orthonormal eigenvectors $\{\boldsymbol{u}_j\}_{j=0}^{n-1} \in \mathbb{R}^{|V|}$ known as the graph Fourier modes, and their associated ordered real nonnegative eigenvalues $\{\lambda_j\}_{j=0}^{n-1}$, known as the frequencies of the graph. The Laplacian is then diagonalized by the Fourier basis $U = [\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{n-1}] \in \mathbb{R}^{|V|\times|V|}$ such that $\mathcal{L} = U\Lambda U^T$, where $\Lambda = \text{diag}([\lambda_0, \ldots, \lambda_{n-1}]) \in \mathbb{R}^{|V|\times|V|}$. The graph Fourier transform of a signal $\boldsymbol{x} \in \mathbb{R}^{|V|}$ is defined as $\hat{\boldsymbol{x}} = U^T\boldsymbol{x}$ and its inverse as $\boldsymbol{x} = U\hat{\boldsymbol{x}}$, in Shuman et al. (2013).

With this definition of graph Fourier transform, one can now define a convolution in the graph domain, $*_G$ of two signals $\boldsymbol{x}$ and $\boldsymbol{y}$ as

$$\boldsymbol{x} *_G \boldsymbol{y} = U\left[(U^T\boldsymbol{x}) \odot (U^T\boldsymbol{y})\right]$$

It follows that a signal $\boldsymbol{x}$ is filtered by g_θ as $\boldsymbol{z} = g_\theta(\mathcal{L})\boldsymbol{x} = g_\theta(U\Lambda U^T)\boldsymbol{x} = Ug_\theta(\Lambda)U^T\boldsymbol{x}$.

Non-parametric filters g_θ offer two problems: they are not localized in space and their learning complexity is $O(|V|)$.

These issues and more were overcome in Defferrard et al. (2016) by approximating the Fourier transform with Chebyshev polynomials, creating the so-called ChebyNets. They considered only parametrized filters of the polynomial Chebyshev form, truncated to order K:

$$g_\theta(\Lambda) = \sum_{k=0}^{K} \theta_k T_k(\tilde{\Lambda})$$

where $\theta \in \mathbb{R}^K$ is a vector of Chebyshev coefficients and $T_k(\tilde{\Lambda}) \in \mathbb{R}^{|V|\times|V|}$ is the Chebyshev polynomial of order k evaluated at $\tilde{\Lambda} = 2\Lambda/\lambda_{\text{max}} - I$, a diagonal matrix of scaled eigenvalues that lie in $[-1, 1]$. Here λ_{max} is the largest eigenvalue of $\mathcal{L}$. The Chebyshev polynomials are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. Note that this approximation makes the filtering operation depend on the K nearest neighbours, making it K localized in space since it's a K^{th} order Chebyshev polynomial in the Laplacian.

Kipf and Welling (2017) took this model and truncated the expansion to $K = 1$, such that the operation is linear on the graph Laplacian spectrum. Additionally they also approximated $\lambda_{\text{max}} = 2$ expecting the rest of the network to adapt to this change in scale. They could now write the filter in it's simple approximated form

$$g_\theta(\Lambda) \approx \theta_0 + \theta_1\tilde{\Lambda} = \theta_0 + \theta_1(\Lambda - I)$$

Furthermore, in order to minimize the multiply operations and number of parameters to learn, they restricted the model with $\theta = \theta_0 = -\theta_1$. Now the filtered signal $\boldsymbol{x}$ can be written as

$$\boldsymbol{z} = g_\theta(\mathcal{L})\boldsymbol{x} = Ug_\theta(\Lambda)U^T\boldsymbol{x} = \theta(2I - U\Lambda U^T)\boldsymbol{x} = \theta(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})\boldsymbol{x}$$

where, in the last step, the relation between the normalized graph Laplacian and its diagonalization $\mathcal{L} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = U\Lambda U^T$ was used. Since $(I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})$ now has eigenvalues in the range $[0, 2]$ they noted that repeated application of this operation would lead to numerical instabilities. Due to this fact, they introduced a *renormalization trick*:

$$I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \to \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}, \text{ with } \hat{A} = A + I \text{ and } \hat{D} \text{ the degree matrix of } \hat{A}.$$

This can be generalized to a multidimensional signal $X \in \mathbb{R}^{|V|\times C}$, where C is the size of each feature vector of X:

$$Z = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}X\Theta$$

This is the final form of pre-activation of the graph convolution layer introduced in Kipf and Welling (2017).

Another explanation for what is going on in Kipf and Welling (2017) is to picture a neural network output of the form $H^{(l+1)} = \sigma(AH^{(l)}W^{(l)})$, where σ is an activation function, $H^{(l)}$ and $H^{(l+1)}$ are the outputs of layers l and $l + 1$ respectively. With this output form, the message in each vertex is not being counted (unless the adjacency matrix has ones in the diagonal, i.e. all vertices are connected to themselves, which is not true in general), so we must make sure to add ones on the diagonal of the adjacency matrix. So we create a new "adjacency matrix" $\hat{A} = A + I$, where I is the identity matrix.

Due to the nature of real-world networks, most of them are scale-free networks and thus have Pareto distributed degrees (the degree being the number of edges in a vertex). This asymmetry in the degrees

will make the feature vectors scale differently, i.e. a node with 100 neighbours will have high values in the feature vector whereas a node with only one neighbor will not. This calls for a normalization of the new adjacency matrix with respect to the degree, i.e. $\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}$, where $\hat{D}$ is the degree matrix, a diagonal matrix with the degrees of each vertex of the new adjacency matrix with the added ones in the diagonal. With the issues resolved, we now have a output function of the form:

$$H^{(l+1)} = \sigma(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H^{(l)}W^{(l)})$$

1.4 The General Perspective

After so many simplifications on the graph convolution operator both perspectives now seem very similar to one another. In Gilmer et al. (2017) they describe the general form of these approaches (and more) as a Message Passing Neural Network (MPNN) in the context of predicting a molecule's properties in quantum chemistry as an alternative to the classical method, with a speedup from the 10^3 seconds of classical discrete Fourier transform to the 10^{-2} seconds of the MPNN. In Xu et al. (2018) they describe how expressive the variants of the graph neural network are in further detail.

2 Results

We first verify the results of the original model in Kipf and Welling (2017) in a semi-supervised setting using the Karate Club dataset from Zachary (1977). Afterwards, we analyze the GCN model with respect to the dependence on the initialization. We then create two-dimensional embeddings with the model and try some new models based on this one, namely adding an attention mechanism and a suggestion made in Kipf and Welling (2017).

2.1 "Free" separations

We used Clauset-Newman-Moore algorithm in Clauset et al. (2004), a modularity clustering algorithm, to find three communities which are displayed as different colors. A first pass of the GCN only sometimes produces a good separation of the data. In Kipf and Welling (2017) they make no mention of how rare this effect really is. In Figure 3 we can see several first passes and it is obvious that not all produce the claimed effects.

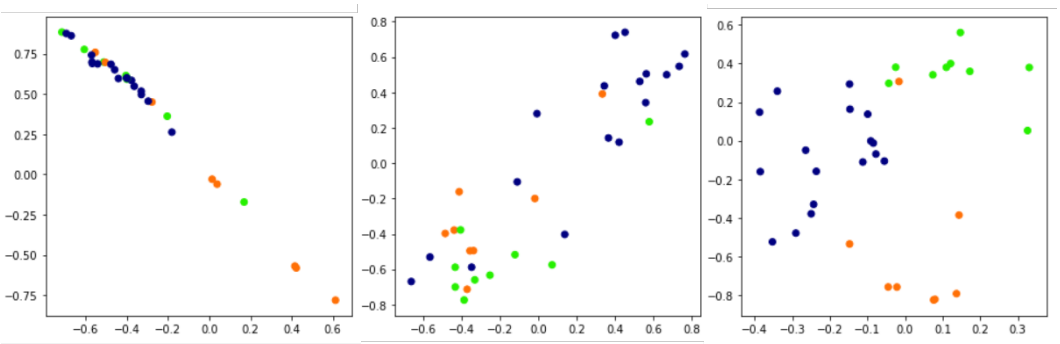

Figure 3: Output of a GCN layer with 3 different sets of random weights. Axes are the 2 dimensions of the embedding of the GCN before the last classification layer. Colors represent community extracted with the Clauset-Newman-Moore algorithm. As can be seen, it is not always that the GCN produces the good results reported in Kipf and Welling (2017). Sometimes, the output points are distributed in a diagonal line (left). The center plot shows a better separation than the left one. The right plot is the kind of plot produced and showed in the original work. By successive trials one is able to obtain it, but it is the exception rather than the rule.

They claim the effect of separating the data is due to a similarity with the Weisfeiler-Lehmann (WL) algorithm from Weisfeiler et al. (1976). The WL algorithm is a simple one:

For all nodes v_i in the graph G:

1. Get features $\{h_{v_j}\}$ of all nodes $\{v_j\}$ in the neighbourhood of v_i;
2. Update node feature $h_{v_i} \leftarrow \mathrm{H}\left(\sum_j h_{v_j}\right)$ where H is an injective hash function.

Repeat until convergence.

The WL algorithm assigns a unique set of features for most graphs. This means that every node is assigned a feature that uniquely describes its role in the graph which is why this algorithm is mostly used to test for graph isomorphism.

The similarity lies both in the way the features are combined and the injectivity of the hash function. Firstly, the WL algorithm selects the neighbours, collects their feature vectors and sums them up, something the GCN also does. Furthermore, both H and σ (the activation function) are injective, assuming no collisions in the hash function. We can see the similarity even more clearly if we write the elementary form of the GCN pass:

$$h_{v_i}^{(l+1)} = \sigma\left(\sum_j \frac{1}{c_{ij}} h_{v_j}^{(l)} W^{(l)}\right)$$

where j indexes the neighbouring nodes of v_i, c_{ij} is a normalization constant from $\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}$, $W^{(l)}$ is the weight matrix of layer l and σ is an activation function. Thus, we are able to get the instantaneous separation of the data without any training, but not as often as one would expect, since the GCN is very chaotic, i.e. it has a big dependence on the initialization of the weights.

2.2 Experiments: analysis and results - Classification Task

In order to properly exam the performance of the GCN model in classification, we conducted three central experiments. Firstly, we tested the model performance on a fixed set of initial weights for a series of labeled nodes. Secondly, we experimented with a fixed set of labeled nodes and varying initial weights. At last, some changes to the structure of the model were carried out. Here we changed the number of layers and the dimension of the weights.

2.2.1 Structure of the model

The following two experiments are made using the same model. Recording the expression of the output of one GCN layer:

$$H^{(l+1)} = \sigma(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}H^{(l)}W^{(l)})$$

we can define the used model as having a first layer with a weight matrix, W, with the number of rows equal to the number of nodes and 5 columns, a second layer having a weight matrix of dimension 5 by 2 and a third layer with a weight matrix of dimension 2 by the number of classes, three in this case. In the end we compute the softmax in the final layer and extract the predicted class from the three possible ones.
For the following experiments we used an Adam optimizer (Kingma and Ba (2014)) with learning rate 0.01 and L2 regularization 5×10^{-4} and Tanh as activation function.

2.2.2 Fixed initial weights and varying labeled nodes for training

The results of this experiment show that the maximum number of correct predictions is significantly affected by the selection of the labeled nodes. In fact, we spotted a difference of 12 on the maximum number of correct predictions for the worst and uncommon case scenario (more results can be seen in the appendix).
It is also noted that there is a point in the training in which the number of corrected predictions reaches a maximum. At this point, training the model for more epochs will over influence the presence of the labeled nodes. This maximum almost always happens when all the labeled nodes are already classified correctly.

2.2.3 Fixed labeled nodes for training and varying initial weights

In this scenario, we performed several experiments with different labeled nodes. All results showed an unstable behaviour. We concluded that the set of initial weights highly affects the performance of the training. For example, in a particular set of labeled nodes the most common maximum number of correct predictions is 29 but the result was different in half of the trained models and ranged from 25 to 31 (for more details on this particular set consult table 1 in the appendix).

2.2.4 Trying different structures

We have already seen how different configurations of the initial weights and labeled nodes affect the training of the model. In this experiment we briefly explored models with different structures. Here we performed changes in the number of layers and the shape of the weights for each layer, meaning, the number of output channels for each layer.
After trying a few different configurations we observed no improvements in the model's predictions comparing to the model used in the previous experiments. In fact, weight initialization had much more impact in the prediction results than the different configurations we tried, making it impossible to extract relevant information about the comparison between different models.

2.3 Embeddings from GCN

The GCN can be used to embed graph structures. For this we used the same training parameters and model of the previous experiments: a 3-layer architecture, with the middle layer with size 2 to embed each node in a 2-dimensional space. The last layer is used for the classification task. These embeddings are obtained by allowing the network to learn how to classify the nodes in a semi-supervised setting, i.e. the network is allowed to compute the loss with only three labeled examples, one for each class. The idea is that the graph structure described by the adjacency matrix allows the network to create embeddings based on the proximity of each node to the labeled nodes.

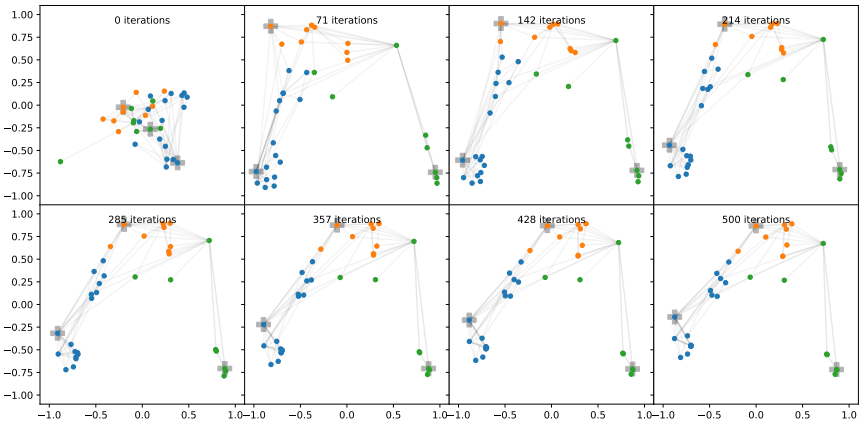

Figure 4: Time evolution of the 2-dimensional embedding of the Karate Club network created by a 3-layer GCN. The training was done in a semi-supervised setting, with only one labeled example per class. For details regarding the evolution of relevant parameters during training consult figure 6 in the appendix section.

Most of the times the 2D embedding successfully separates the different classes. However, in some training configurations, the 2D embedding presented the nodes in a straight line, which may lead to the conclusion that the model didn't take advantage from a second dimension.

2.4 Embeddings from GCN with Attention

To try to build on top of the work of Kipf and Welling (2017), we decided to experiment with adding an attention mechanism to the GCN. We did this by adding some free parameters to the adjacency matrix, such that the update rule of the network is of the form:

$$H^{(l+1)} = \sigma(B^{(l)}\hat{A}H^{(l)}W^{(l)})$$

The idea was to have this new matrix $B^{(l)}$ modify the weights of the different edges between nodes. This comes with additional problems. Though we're adding expressive power to the model, we now have to train many more parameters, as many as there are edges. We also lose the power of numerical stability added to the model by normalizing the adjacency matrix, since now that normalization is now made irrelevant since the new parameters are learned during training.

We trained a 3-layer GCN model with a middle layer with size 2, so as to produce two-dimensional embeddings. The last layer had size 3, so it could be used to compute a cross entropy loss with respect to the 3 classes obtained previously through modularity-based community finding. The actual loss was computed only with three randomly assigned nodes, with only one node per class, i.e. in a semi-supervised setting. An Adam optimizer was used with learning rate 0.01 and L2 regularization 5×10^{-4}. The approach was once again to feed the network the identity matrix, therefore taking an implicitly feature-less approach. We used the Tanh as activation function.

What we found was that indeed this model experienced numerical instabilities with some gradient explosions happening sporadically. This makes it so that the model performs poorly when producing embeddings and even in the classification task. The produced embeddings and relevant training parameters can be consulted in figures 10 and 11 of the appendix section.

In addition to the previous experiment, several other approaches involving the idea of an attention mechanism were tried. Since in our first approach we didn't normalize the matrix $B^{(l)}\hat{A}$, our second approach was to combine the normalization with the update rule, therefore producing the following expression:

$$H^{(l+1)} = \sigma(\tilde{D}^{-1/2}B^{(l)}\hat{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)})$$

where $\tilde{D}$ is the degree matrix of the attentive adjacency matrix $B^{(l)}A$.
After implementing this solution, no improvements were observed on the numerical stability nor the quality of the embeddings and the accuracy of the model.
Another approach was fixing the attention matrix across all layers such that, for all l, $B^{(l)} = B$. The idea was that there would be less parameters and the structure of the adjacency matrix wouldn't be so overshadowed by the added model complexity. However, after implementation, this approach didn't produce better results and the numerical instabilities did not go away. In fact, exploding gradients were still common during training.

2.5 Embeddings from IGCN

In the work of Kipf and Welling (2017), they underline several limitations of the original model and leave it as future work. One of these limitations is the fact that when we add the identity matrix to the adjacency matrix to allow the messages on each node to have an impact on the next message along with the messages of each neighbour, we are assuming equal importance. Therefore, they suggest work to be done on including a trade-off parameter λ to be learned, in the definition of $\hat{A}$:

$$\hat{A} = A + \lambda I$$

We tried this new model definition, henceforth refered to as IGCN, for importance graph convolution network. We used this new model as a layer in a 3-layer model. The middle layer was once again of size 2 to produce 2-dimensional embeddings. The last layer had size 3, to correspond to the three classes obtained from the previously found modularity-based communities. We used an Adam optimizer with learning rate 0.01 and L2 regularization 5×10^{-4}. As before the cross entropy loss was computed with a single labeled example per class, in a semi-supervised setting. We used the Tanh as activation function.

Comparing the embeddings of the IGCN model (figure 12 of the appendix section) to the original model, we can see a clear improvement, even in the classification task.
Developing this idea further, we tried two more approaches. In the previous approach we computed

the degree matrix, $\tilde{D}$, of the adjacency matrix, A. Our second approach was to compute the degree matrix of the adjacency matrix with the trade-off parameter, $\hat{A} = A + \lambda I$, but no better accuracy nor quality of the embeddings were observed.
At last, we tried to give each element of the diagonal of the adjacency matrix a different learned parameter, extending the trade-off parameter from a scalar to a vector with the size equal to the number of nodes. In this way we hoped to increase the expressive power of the model by giving each node a unique trade-off parameter. Once again, the numerical instabilities in the training procedure made this model perform very poorly, decreasing the accuracy and the quality of the embeddings.

3 Classification on Citeseer dataset

In Kipf and Welling (2017), they compared the GCN model with several other models as baselines, on different network datasets. We replicated their work on the Citeseer dataset (Giles et al. (1998)), on which they obtained a 70.3% accuracy using only 3.6% label rate, i.e. using only 3.6% of the entire network as labeled examples. Citeseer is a citation network, with 3,327 nodes, 4,732 edges, 6 classes and 3,703 features per node. The features are sparse bag-of-words vectors of each document.

We obtained similar results, a 69.7% accuracy with the same label rate. We used a 2-layer GCN with ReLU activation. We used Adam with learning rate 0.01 and L2 regularization 5×10^{-4}. The produced embedding can be consulted in figure 13 of the appendix section.

4 Conclusions

We provided a historical view on the evolution of the graph neural network and explained how both the recurrent view and the convolutional view converged over time due to iterative approximations. A thorough explanation of the theory behind the graph convolutional network (GCN) in Kipf and Welling (2017) was put forth, combined with earlier works. An analysis of the dependence on the setting of the initial weights of the GCN was done, and concluded that this model is very prone to instabilities depending on the initial conditions. We saw how to create embeddings of the nodes of graphs via an embedding layer of any size and tried to extend this model with attention and then with the suggested change (suggested in Kipf and Welling (2017)) of adding a trade-off parameter between supervised and unsupervised learning. We concluded that the attention model we idealized had many problems and failed to produce any useful embeddings or classifications. However, the trade-off parameter allowed for cleaner embeddings and increased the accuracy in the Karate Club dataset. Lastly, we verified the accuracy on a specific dataset (Citeseer) in Kipf and Welling (2017). We also embedded the Citeseer dataset, to demonstrate that it is possible to embed large networks in 2-dimensional space and still get some differentiation of the data. This shows that if we increase the dimensions of the embedding, we get something which is indeed useful for classification, even if we lose the ability to visualize it.

References

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *In NIPS, volume 14, pages 585–591*, 2001.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203 [cs.LG]*, 2013.

Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *arXiv:cond-mat/0408187 [cond-mat.stat-mech]*, 2004.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Neural Information Processing Systems*, 2016.

C. Lee Giles, Kurt D. Bollacker, and Steve Lawrenc. Citeseer: an automatic citation indexing system. *Digital Libraries 98 - Third ACM Conference on Digital Libraries*, 1998.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *International Conference on Machine Learning*, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980 [cs.LG]*, 2014.

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

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *International Conference on Learning Representations*, 2016.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.

D. Shuman, S. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine, 30(3):83–98*, 2013.

Boris Weisfeiler, A. Lehman, G. M. Adelson-Velsky, V. Arlazarov, I. Faragev, A. Uskov, I. Zuev, and M. Rosenfeld. On construction and identification of graphs. *volume 558 of Lecture Notes in Mathematics. Springer-Verlag, Berlin*, 1976.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *International Conference on Learning Representations*, 2018.

Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Anthro. Research 33(4), 452-473*, 1977.

5 Appendix

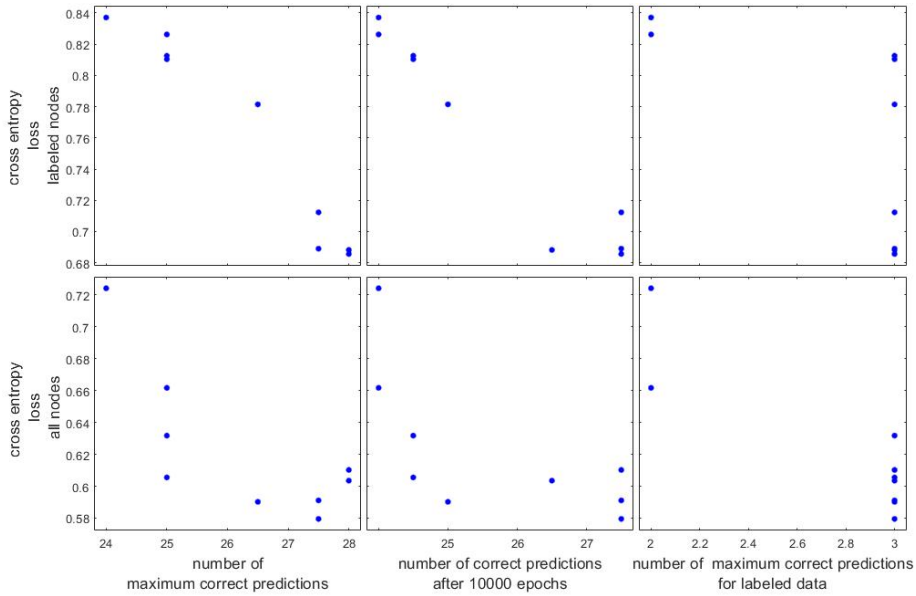

Figure 5: Set of graphs representing the relationship between different training outcomes. Each point represents a different set of labeled nodes used for training.

Maximum number of correct predictions	Number of corresponding cases
25	1
28	2
29	7
30	3
31	1

Table 1: Maximum number of correct predictions and corresponding number of training trials involving different initial weights for a total of 14 training trials.

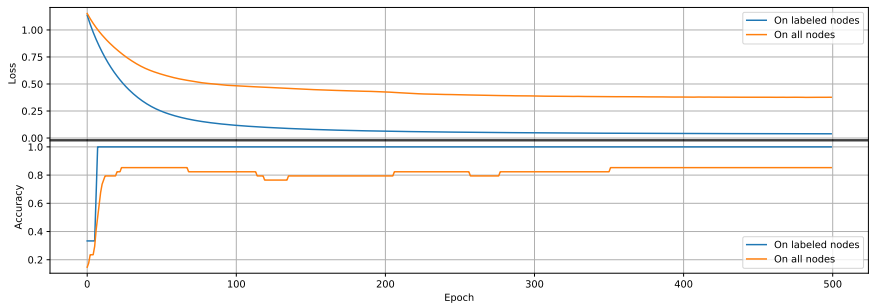

Figure 6: Training statistics of the GCN on the Karate Club network. Cross entropy loss on the labeled nodes and on all nodes (top). Accuracy on the labeled nodes and on all nodes (bottom).

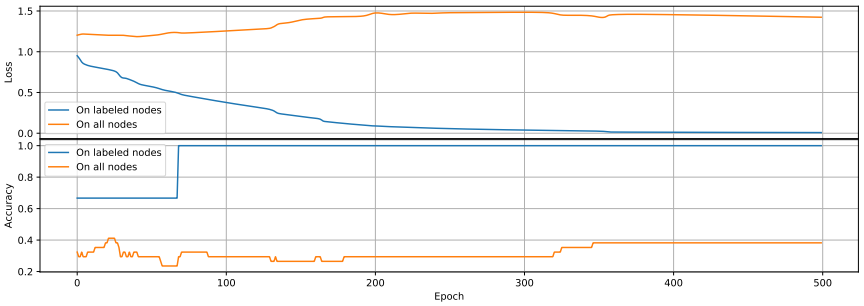

Figure 7: Training statistics of the attention GCN on the Karate Club network. Cross entropy loss on the labeled nodes and on all nodes (top). Accuracy on the labeled nodes and on all nodes (bottom).

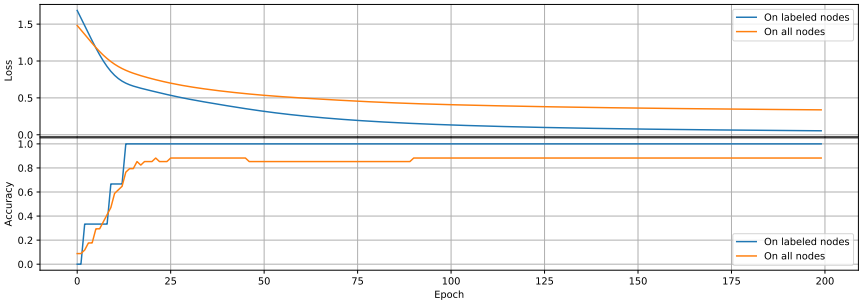

Figure 8: Training statistics of the IGCN on the Karate Club network. Cross entropy loss on the labeled nodes and on all nodes (top). Accuracy on the labeled nodes and on all nodes (bottom).

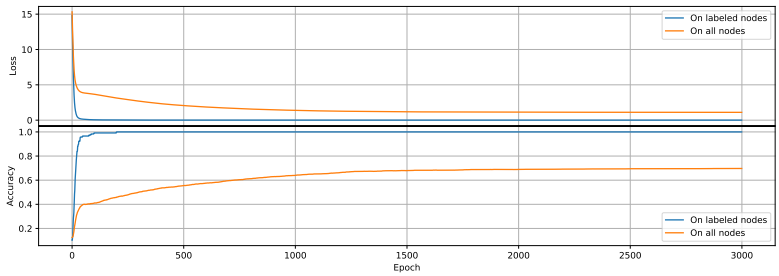

Figure 9: Training statistics of the GCN on the Citeseer network. Cross entropy loss on the labeled nodes and on all nodes (top). Accuracy on the labeled nodes and on all nodes (bottom).

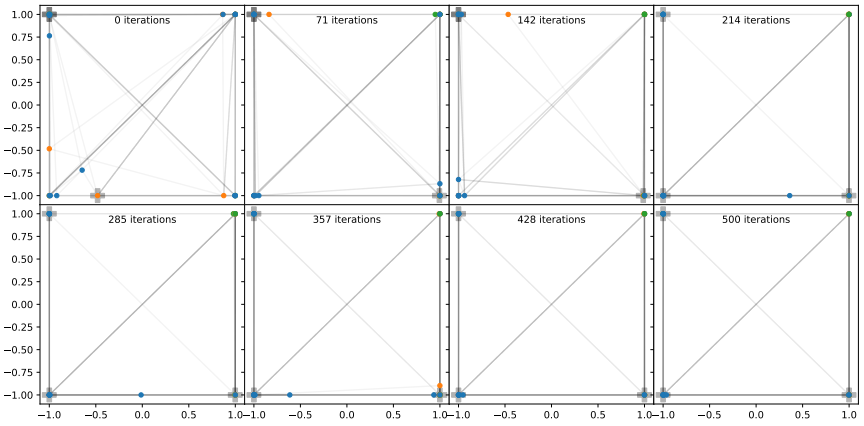

Figure 10: Time evolution of the 2-dimensional embedding of the Karate Club network created by a 3-layer GCN with attention. The training was done in a semi-supervised setting, with only one labeled example per class.

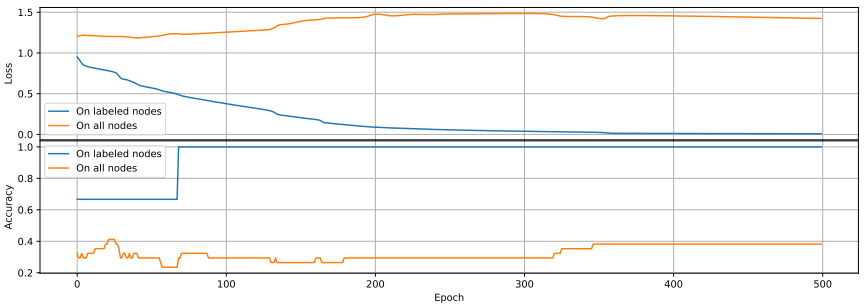

Figure 11: Training statistics of the attention GCN on the Karate Club network. Cross entropy loss on the labeled nodes and on all nodes (top). Accuracy on the labeled nodes and on all nodes (bottom).

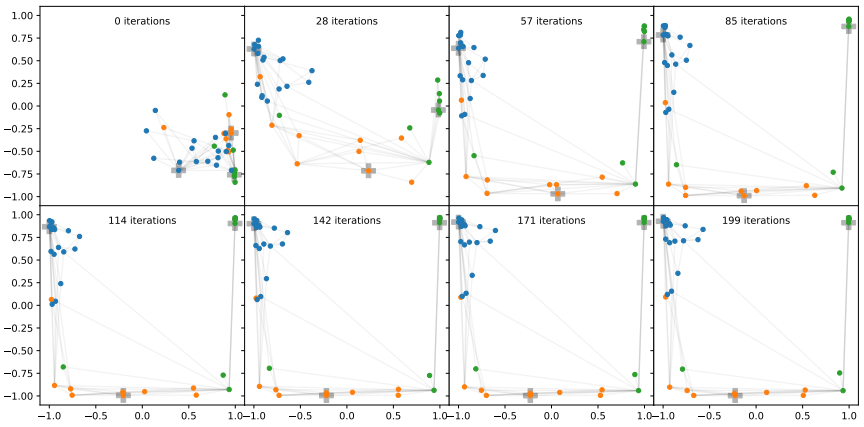

Figure 12: Time evolution of the 2-dimensional embedding of the Karate Club network created by a 3-layer IGCN. The training was done in a semi-supervised setting, with only one label per class. For details regarding the evolution of relevant parameters during training consult figure 8 in the appendix section.

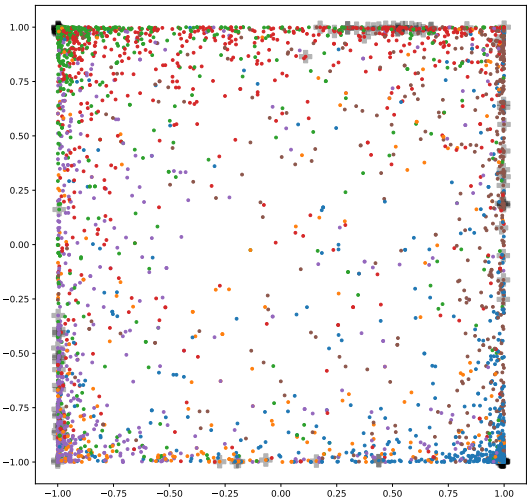

Figure 13: Two-dimensional embedding of the entire Citeseer dataset. While it may look messy, we can still see a dominance of the same colors in each corner of the plot. This means an embedding is being created but that we need to increase the dimensionality for it to become closer to linearly separable.