Semi-Supervised Classification of Graph Nodes using Exponential Decay
L42: Assessment 2

Jan Ondras (jo356), Trinity College
Word count: 2 497 words*

February 23, 2018

Abstract

Recently, as we deal with large complex networks in various domains such as biology, sociology or computer science, semi-supervised learning on graph-structured data has attained lots of attention.

In this report I propose a new approach (MLP_λ) to incorporate relationships between graph nodes in a semi-supervised classification problem, using the multilayer perceptron model. In particular, new features of a node are calculated as a sum of contributions from all other nodes exponentially weighted by their shortest path distance from the node. I also investigated the effect of various feature scaling methods and determined the optimal decay parameter λ for given datasets.

The proposed method was evaluated on three citation networks (*Cora*, *Citeseer*, *PubMed*) and compared to the baseline multilayer perceptron (when training examples were treated as independent) and other previous studies. The results show that my approach reaches the performance of most earlier methods, however, it falls behind the recent success of Graph Convolutional Networks (GCNs) or Graph Attention Networks (GATs).

Contents

*this word count was computed by `texcount -inc report.tex`

Overview

This report is structured as follows. Section 1 defines the task being solved and lists the related work in the field. Section 2 describes the details of the proposed approach along with its pros and cons. The performed experiments and obtained results are presented and analysed in Section 3. Section 4 concludes the report and discusses future research directions.

1 Introduction

Nowadays, when many large datasets come in a form of graphs, leveraging the graph structure becomes crucial with applications in many domains such as cortical mesh segmentation, molecular fingerprinting or modelling of multi-agent interactions.

1.1 Problem statement

In this project I focus on semi-supervised graph node classification where the labels are available only for a small subset of nodes. This task can be phrased as follows.

- **Input:** a matrix of node features, $\mathbf{X} \in \mathbb{R}^{N \times F}$, with F features for each of the N nodes; partitioning of graph nodes into train, validation, and test sets; node labels for training; and a graph adjacency matrix, $\mathbf{A} \in \mathbb{R}^{N \times N}$, such that

$$A_{ij} = \begin{cases} 1 & \text{if nodes i and j are related,} \\ 0 & \text{otherwise.} \end{cases}$$

  We can notice that $\mathbf{A} = \mathbf{I}$ means that every node is an independent training example. We will further assume that all edges are undirected and unweighted.

- **Output:** a matrix of node class probabilities, $\mathbf{Y} \in \mathbb{R}^{N \times K}$, where $Y_{ij} = P(\text{node } i \in \text{class } j)$ and K is the number of classes.

In general, there are two common types of learning on graphs:

1.) **Transductive learning** – the whole feature matrix $\mathbf{X}$ (including test nodes) is exposed during the training.

2.) **Inductive learning** – not all nodes are exposed in advance. For instance, if test nodes are gradually inserted into the graph or if the trained model is tested on a completely unseen test graph. This is a much more challenging task.

Further, I will primarily address the transductive learning problem.

1.2 Related work

Previous studies usually use a per-node classifier and based on the incorporation of graph structure they can be split into two main categories:

1.) **Indirect graph structure incorporation**

One approach to reinforce the graph structure is to constrain the learnt features based on graph edges. The work *semi-supervised embedding* by Weston et al. [1] achieves this by augmenting the loss function with a similarity constraint, assuming that edges represent node similarity and that connected nodes are likely to have the same label. This is often referred to as graph-based regularisation [2, 3, 4].

An alternative approach is to augment the input layer with graph-structure features. For example, the method *Deepwalk* [5] uses random walks to analyse the graph structure and to extract structural features for each node. These features are then concatenated with original node features. This

approach was later improved by *LINE* [6] and *node2vec* [7], preserving the main idea. The advantage of these methods is that they can deal with fully unsupervised learning problems. Another method, *Planetoid* [8], extends the idea of *DeepWalk* for cases when labels are available. It consists of two phases: sampling based on random walks and sampling based on labels. An adaptation of Planetoid can also address inductive learning problems.

2.) **Explicit graph structure incorporation**

These methods compute a completely new representation $\mathbf{h_i}$ for each node i based on its original features and graph structure. The representation $\mathbf{h_i}$ is then used to classify each node independently. For instance, Graph Neural Networks [9, 10] compute the representations $\mathbf{h_i}$ by an iterative process that propagates node states until convergence. The nodes are then classified based on their converged states. Li et al. [11] further improve this idea by proposing gated recurrent units [12] in the propagation step, resulting in Gated Graph Neural Networks.

Instead of a "time-domain" propagations, more recent research attempts to generalise convolutional operations to graphs. Progress in this direction can be further divided into:

(a) **Spectral approaches**

They aim to take the computational advantage of operating in the spectral domain and use a graph Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{A}$ (where $\mathbf{D}$ is the diagonal degree matrix with $D_{ii} = deg(i)$). Using the spectral decomposition of $\mathbf{L}$, one can enter the spectral domain of the graph where the convolution becomes point-wise multiplication[1]. One example is the work Graph Fourier Transform [13] by Bruna et al. However, computing the spectral decomposition of $\mathbf{L}$ is expensive ($O(N^3)$) and also the use of Fourier Transforms means that the localisation property is lost. These drawbacks were addressed by Chebyshev networks [14] that employ a family of Chebyshev polynomials rather than computing the Fourier Transform. This simplifies the computation since these polynomials have a recursive definition and also the localisation property is restored. Finally, Graph Convolutional Networks (GCNs) simplify the Chebyshev framework by restricting the filters to operate in a 1-step neighborhood around each node which improves both predictive performance on small training sets and computational performance on larger graphs. However, the major drawback of Chebyshev networks, that different nodes in a neighbourhood share the same weights, is still present. In general, the fundamental limitation of spectral-based approaches is that they are inappropriate for arbitrary inductive problems as they rely on a fixed graph Laplacian.

(b) **Non-pectral approaches**

These methods define convolutions on graphs directly, operating on sets of spatially close neighbours [15, 16, 17]. The most recent and successful approach in this area is Graph Attention Networks (GATs) [18] that use masked self-attentional layers which enables having different weights for nodes in a neighborhood, without a need for expensive matrix operations or dependence on upfront knowledge of the graph structure. This approach yields impressive performance across several large-scale inductive benchmarks as well as on transductive problems.

2 Methods

2.1 Baseline MLP

A baseline suitable for comparison of methods can be obtained by completely ignoring the graph structure and treating each node as an independent training example. The classification problem then becomes a classical supervised learning which can be addressed using Multilayer perceptron (MLP) model.

[1] by the convolution theorem

2.2 Proposed MLP$_\lambda$

2.2.1 Theory

My proposed method MLP$_\lambda$ tries to incorporate the graph structure by creating a new feature matrix $\hat{\mathbf{X}} \in \mathbb{R}^{N \times F}$, that, for each node, captures features from nodes distant any number of hops.

The two main assumptions being made are:

- closer nodes in terms of the shortest path length (SPL) are more likely to have the same label and more similar features than more distant nodes,
- closer nodes affect their new feature vectors more than more distant nodes.

To prevent the dependence between the rank of the new feature matrix $\hat{\mathbf{X}}$ and the graph structure I use an additive approach, and to address the second assumption I employ an exponential weighting scheme so that the contribution to new feature vectors falls down exponentially with the SPL between nodes.

In particular, the new feature vector $\hat{\mathbf{x}}_i \in \mathbb{R}^F$ of node i is calculated as

$$\hat{\mathbf{x}}_i = \sum_{j=1}^{N} \mathbf{x}_j e^{-\frac{L(i,j)}{\lambda}} \tag{1}$$

where $\mathbf{x}_j$ is the original feature vector of node j, $L(i,j)$ is the SPL between nodes i and j, and λ is the decay parameter that controls how quickly the contribution falls down with the SPL between nodes, namely, the contribution from a node distant L is $e^{1/\lambda}$-times smaller then from a node distant $L-1$. If there is no path between i and j the length $L(i,j)$ is set to ∞ which ensures that disconnected nodes do not affect each others new features.

The new feature matrix $\hat{\mathbf{X}}$ can be then compactly expressed as

$$\hat{\mathbf{X}} = \mathbf{\Lambda}_\lambda \mathbf{X} \tag{2}$$

where the weight matrix $\mathbf{\Lambda}_\lambda \in \mathbb{R}^{N \times N}$ has entries $\Lambda_{\lambda ij} = e^{-\frac{L(i,j)}{\lambda}}$.

Let us now investigate the following two limit cases:

- $\lambda \to 0$
  This means that features of other nodes are completely ignored ($\mathbf{\Lambda}_0 = \mathbf{I}$) and so the new feature matrix becomes
  $$\hat{\mathbf{X}} = \mathbf{X}$$
  which is equivalent to the Baseline MLP. In other words, Baseline MLP $\Leftrightarrow$ MLP$_{\lambda=0}$.
- $\lambda \to \infty$
  This means that nodes of each connected component C of the graph have identical new feature vectors such that
  $$\forall i,k \in C. \quad \hat{\mathbf{x}}_i = \hat{\mathbf{x}}_k = \sum_{j=1}^{|C|} \mathbf{x}_j$$
  which represents the other extreme (of graph structure incorporation) where nodes within a connected component become indistinguishable in terms of their new features.

Next, we can investigate what range of values can new features take. If the original feature matrix $\mathbf{X}$ was binary, then the new feature matrix $\hat{\mathbf{X}}$ will contain features with magnitudes bounded by

$$\forall i,f. \quad 0 \leq \hat{X}_{if} \leq 1 + (N-1)e^{-\frac{1}{\lambda}} \tag{3}$$

where the upper bound is obtained by considering a star graph. We can see that the higher the decay parameter λ is, the higher values can new features take.

2.2.2 Complexity

The space complexity of this approach does not exceed a constant factor of the Baseline MLP. This holds even for sparse adjacency matrices, since Eq. (1) allows us to compute new feature vectors one-by-one requiring us to store only the SPLs for a current node, thus avoiding the storage of the whole weight matrix $\mathbf{\Lambda}_\lambda$.

However, there is a difference in time complexity which in the case of MLP_λ is dominated by two operations:

- Calculation of all pairs SPLs which can be done in $O(N^2 \log N)$ time using Johnson's algorithm [19].
- Computation of all new feature vectors (Eq. (1)) which is $O(N^2 F)$.

Hence, the total time complexity of the feature preprocessing step is $O(N^2(F + \log N))$.

2.2.3 Advantages and disadvantages

I identified the following pros and cons of my proposed method before any experiments were performed:
Advantages

- Captures relationships between *arbitrarily* distant nodes.
- Correctly handles disconnected graphs.
- Does not increase dimensionality of feature vectors.
- Differentiates between various number of nodes at a particular SPL distance.
- Can be naturally applied to weighted and directed graphs (contrary to GCNs that are limited to undirected graphs).

Disadvantages

- Unable to handle edge features (similarly as GCNs).
- Expensive computation of new feature matrix.

2.3 Implementation

I implemented both Baseline MLP and MLP_λ in Python using the machine learning library Keras [20].

3 Experiments, results and analysis

3.1 Datasets

I evaluated my implementations of Baseline MLP and MLP_λ on three citation networks: *Cora*, *Citeseer*, and *PubMed*. Here, the node classification corresponds to topic classification where each node is a paper represented by bag-of-words binary features and each edge is a citation. The parameters of these datasets are summarised in Tab. 1 and their graphs can be found in Appendix A, Fig. 5.

Table 1: Dataset statistics, as retrieved from GCN repository [2].

Dataset	#nodes (N)	#edges (E)	#features (F)	#classes (K)	#training nodes	#validation nodes	#test nodes
Cora	2 708	5 278	1 433	7	140 (5.2%)	500 (18.5%)	1000 (36.9%)
Citeseer	3 327	4 676	3 703	6	120 (3.6%)	500 (15.0%)	1000 (30.0%)
PubMed	19 717	44 327	500	3	60 (0.3%)	500 (2.5%)	1000 (5.1%)

[2] https://github.com/tkipf/gcn

3.2 Baseline MLP

Since all three datasets have binary features there was no need for feature scaling. I trained the Baseline MLP with hl hidden layers containing hu hidden units each for a maximum of 10 000 epochs with early stopping with a window size of 10, i.e. I stopped the training if the validation accuracy did not increase for 10 consecutive epochs. The cross-entropy loss was optimised using Adam optimiser [21] and the whole training set was used for every iteration, i.e. every epoch had one step.

For hidden layers I used a rectified linear unit (ReLU) activation function and He initialisation of weights as suggested by [22]. After each hidden layer I applied a dropout of 0.5 to prevent from overfitting. The final layer with K units used the *softmax* function to target K classes.

For each dataset, the hyperparameters $hl \in \{1, 2, 3, \dots 10\}$ and $hu = 5i$ with $i \in \{2, 3, 4, \dots 16\}$ were determined by cross-validation (only a single split since the validation set was fixed). For every combination of these hyperparameters, 100 training and validation runs with random weight initialisations were performed and mean validation accuracy was reported. The heatmaps from cross-validation are shown in Fig. 1 and the best hyperparameters for each dataset are summarised in Tab. 2.

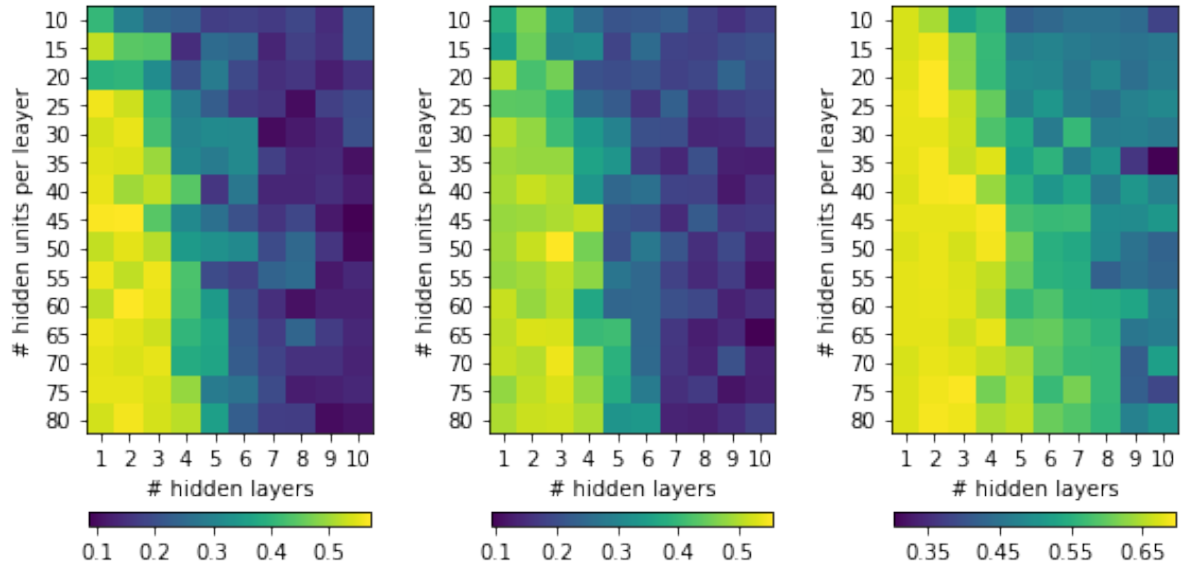

Figure 1: Mean validation accuracy in terms of hyperparameters of the Baseline MLP architecture, for three citation networks: *Cora*, *Citeseer*, and *PubMed* (from left to right).

Table 2: Optimal parameters found for Baseline MLP.

Dataset	#hidden layers (hl)	#hidden units per layer (hu)
Cora	2	60
Citeseer	3	50
PubMed	2	20

For each dataset and its corresponding best hyperparameters (hl, hu) I trained and tested my models over 100 runs with random weight initialisations and I report the mean testing accuracy along with the standard deviation. The results are summarised later in Tab. 3.

3.3 Proposed MLP$_\lambda$

I followed the same[3] experimental set-up as for the Baseline MLP with the only difference that I did not tune the hyperparameters (hl, hu) and for each dataset I used the best architecture according to Tab. 2.

In my experiments, I investigated

- the effect of feature scaling of the new feature set,
- and the choice of decay parameter λ.

3.3.1 Feature scaling

In the case of MLP$_\lambda$ the features are no longer binary and so feature scaling might be appropriate. Thus, for the Cora dataset I compared the following three feature scaling methods on the validation set:

- **No scaling** – features are bounded according to Eq. (3).
- **MinMax scaling** – features are rescaled to the interval $[0, 1]$ along each dimension independently.
- **Standard scaling** – features are standardised by removing the mean and scaling to unit variance, along each dimension independently.

It is important to note that the parameters of the latter two scaling methods were determined on the training set only, and consequently the transformation was applied to all (training, validation, and test) sets.

The results in terms of validation accuracy and standard deviation (over 100 runs) are shown in Fig. 2, for a range of decay parameters $\lambda \in S_\lambda$ where $S_\lambda = \{0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 3.5, 4, 4.5, 5, 5.5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 35, 40, 45, 50, 100, 200, 300, 400, 500, 1000\}$.

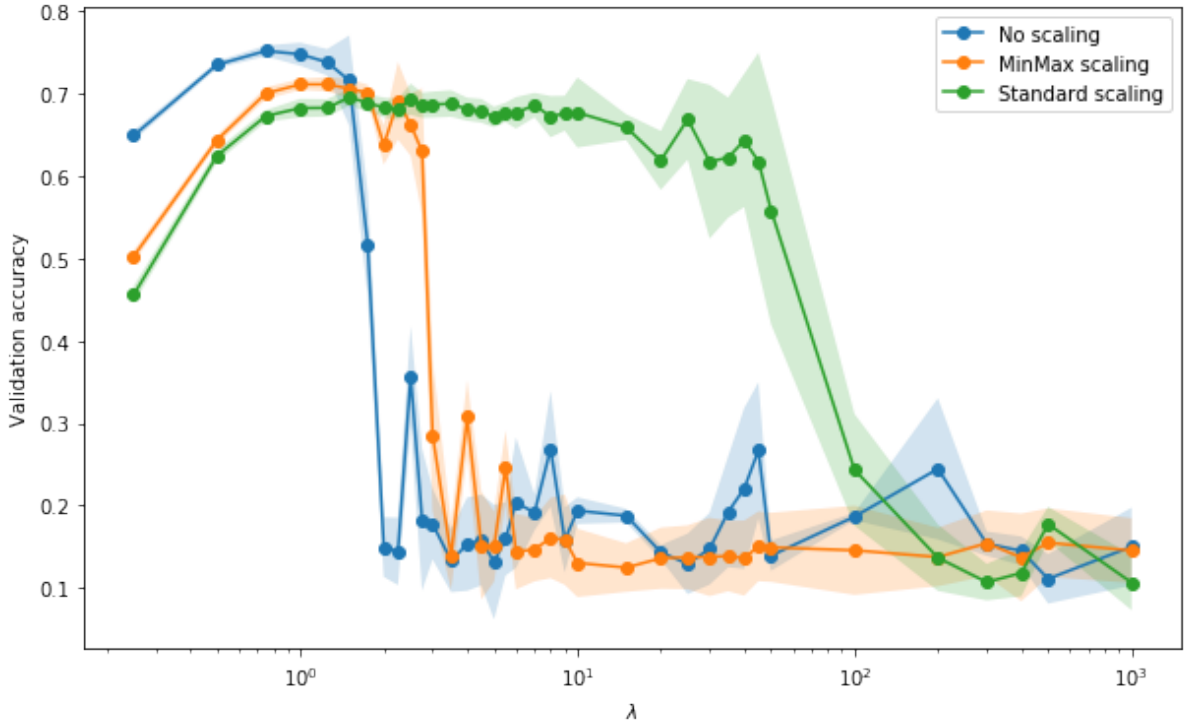

Figure 2: Mean validation accuracy ± standard deviation for various feature scaling methods and decay parameters $\lambda \in S_\lambda$, evaluated on the *Cora* citation network.

[3] In terms of the activation functions, optimiser, loss, #epochs, early-stopping, and number of runs with random weight initialisations.

We can see that the *No scaling* method achieves higher validation accuracy than the other two methods for a particular values of λ, namely, for $0.25 \leq \lambda \leq 1.5$. This suggests two things: (i) some values of λ can provide better performance than others; and (ii) in this case feature scaling seems to throw away some important information and worsen the performance. Therefore, for the following experiments I used the *No scaling* method.

3.3.2 Choice of decay parameter λ

For each dataset, I trained and tested the MLP_λ model[4] for various values of the decay parameter $\lambda \in S_\lambda$. The results (averaged over 100 runs) are depicted in Fig. 3. We can observe that for each dataset there is a value of λ that maximises the classification performance, and interestingly enough, the optimal value for each dataset is the same, $\lambda = 0.75$. Thus, I can conclude that the best performing MLP_λ model on these datasets is $\text{MLP}_{\lambda=0.75}$. Furthermore, these results suggest that the value $\lambda = 0.75$ might be the best option in general, however, to make such a conclusion, experiments on more datasets would have to be performed. The best testing accuracies for each dataset are later summarised in Tab. 3.

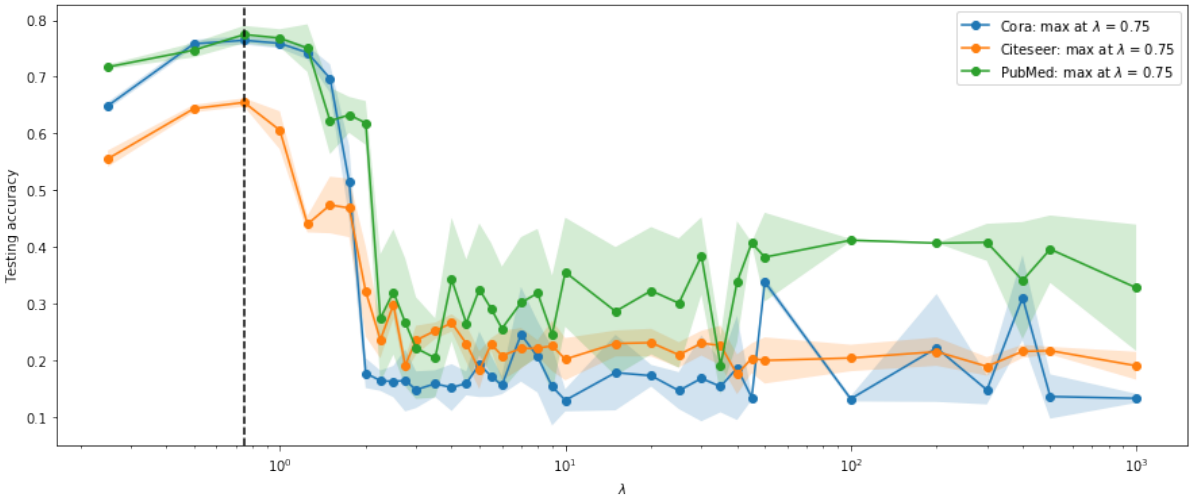

Figure 3: Mean testing accuracy $\pm$ standard deviation for various decay parameters $\lambda \in S_\lambda$, evaluated on three citation networks: *Cora*, *Citeseer*, and *PubMed*.

For the best choice $\lambda = 0.75$, we can visualise exponential feature weights $w_{Sj} = e^{-L(S,j)/0.75}$ of nodes (j) from a sample connected component from *Cora* network as they affect the new feature vector $\hat{\mathbf{x}}_S$ of a particular node S in this component. This is illustrated by Fig. 4 (left). Also, we can look at the total contribution $c_S(L)$ to node S's features from all nodes at a particular distance L from S, namely,

$$c_S(L) = \frac{N_L e^{-L/0.75}}{\sum_{L'=0}^{\infty} N_{L'} e^{-L'/0.75}}$$

where N_L is the number of nodes at the SPL L from node S. Fig. 4 (right) clearly shows that for this graph component nodes distant one hop from node S significantly affect its new feature vector $\hat{\mathbf{x}}_S$ and even surpass the contribution from node S's original feature vector.

[4]With architectures given by Tab. 2.

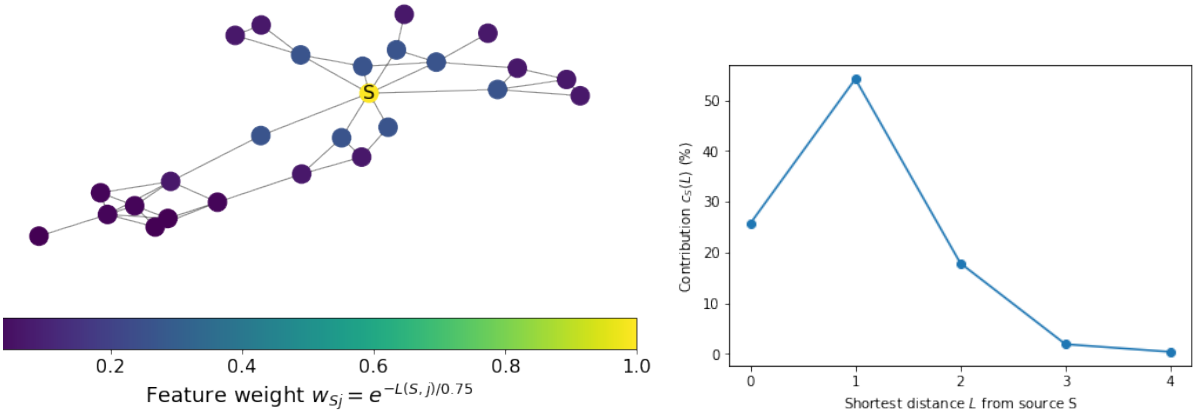

Figure 4: *Left:* exponential feature weights w_{Sj} of nodes j from a sample connected component from *Cora* dataset for $\lambda = 0.75$, affecting the new feature vector $\hat{\mathbf{x}}_S$ of node S. *Right:* contribution to node S's features from all nodes at a particular shortest path distance from node S, scaled so that these contributions sum to 100%.

3.4 Performance comparison

Finally, I compared the performance of my best model $\text{MLP}_{\lambda=0.75}$ with other approaches, as shown in Tab. 3 (results for other methods were taken from [23] and [18]).

Table 3: Testing accuracy (%) with standard deviation of various methods on three citation networks.

Method	Cora	Citeseer	PubMed
Chance level[5]	14.3	16.7	33.3
Baseline MLP	55.4 ± 0.6	50.8 ± 0.7	71.4 ± 2.6
MLP$_{\lambda=0.75}$	76.4 ± 0.7	65.5 ± 0.7	77.5 ± 1.5
ManiReg [4]	59.5	60.1	70.7
SemiEmb [1]	59.0	59.6	71.1
LP [2]	68.0	45.3	63.0
DeepWalk [5]	67.2	43.2	65.3
ICA [24]	75.1	69.1	73.9
Planetoid* [8]	75.7	64.7	77.2
Chebyshev [14]	81.2	69.8	74.4
GCN [23]	81.5	70.3	**79.0**
GAT [18]	**83.0** $\pm$ 0.7	**72.5** $\pm$ 0.7	**79.0** $\pm$ 0.3

As we can see, $\text{MLP}_{\lambda=0.75}$ clearly outperforms the Baseline MLP and achieves comparable performance with earlier approaches on all three datasets, in particular, it closely matches the results of Planetoid*. However, it falls behind the performance of GCN and GAT on all three datasets. We can also notice that even though the Chebyshev method outperforms $\text{MLP}_{\lambda=0.75}$ on *Cora* and *Citeseer*, it does not so on *PubMed*. This suggest that my approach might be suitable for large-scale networks.

[5]Expected accuracy if classes are assigned by random guessing, in this case, $1/K$.

4 Conclusion and Future Work

4.1 Conclusion

This work presents a novel method MLP_λ for incorporating the graph structure in machine learning problems, by creating an amended feature set where new feature vectors are calculated as a weighted sum of feature vectors of nodes from the same connected component of the graph. These weights decay exponentially with the shortest path length between nodes.

My experiments on three citation networks (*Cora*, *Citeseer*, *PubMed*) show that the optimal decay parameter is $\lambda = 0.75$ and that the feature scaling is not appropriate in this case. When compared to other earlier methods the $\text{MLP}_{\lambda=0.75}$ model competes with the best ones, however, it is outperformed by recent Graph Convolution Networks and Graph Attention Networks by a significant margin.

4.2 Future work

It seems that future research in this area will focus on non-spectral methods and inductive problems.

Nevertheless, regarding the method MLP_λ, other datasets similar to *PubMed* could be tested to investigate its suitability for large-scale networks as suggested in Sec. 3.4.

Also, it would be interesting to extend MLP_λ to handle edge features. One possibility would be to insert auxiliary nodes that would represent edge features in the original graph.

A Citation networks for evaluation

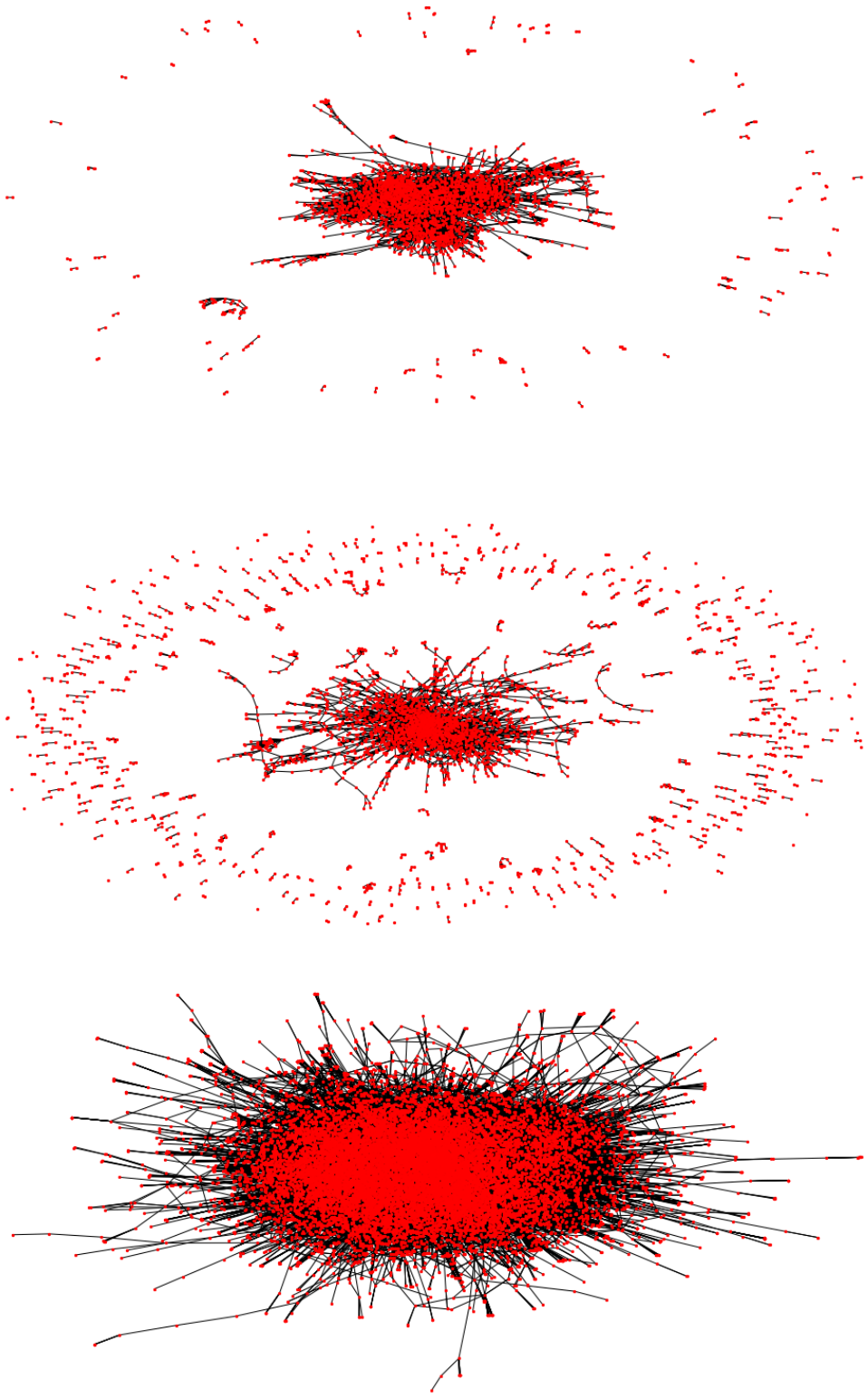

Figure 5: Graphs of citation networks used for evaluation: *Cora*, *Citeseer*, and *PubMed* (from top to bottom).

References

[1] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[2] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003.

[3] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328, 2004.

[4] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(Nov):2399–2434, 2006.

[5] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[6] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[8] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. *arXiv preprint arXiv:1603.08861*, 2016.

[9] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pages 729–734. IEEE, 2005.

[10] 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.

[11] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[12] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

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

[14] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[15] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[16] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016.

[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

[18] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

[19] Donald B Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)*, 24(1):1–13, 1977.

[20] François Chollet et al. Keras. `https://github.com/keras-team/keras`, 2015.

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

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[23] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[24] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 496–503, 2003.