Label Propagation with Neural Networks

Aditya Pal
Pinterest
aditya.pal@gmail.com

Deepayan Chakrabarti
University of Texas, Austin
deepay@utexas.edu

ABSTRACT

Label Propagation (LP) is a popular transductive learning method for very large datasets, in part due to its simplicity and ability to parallelize. However, it has limited ability to handle node features, and its accuracy can be sensitive to the number of iterations. We propose an algorithm called LPNN that solves these problems by a *loose-coupling* of LP with a feature-based classifier. We experimentally establish the effectiveness of LPNN.

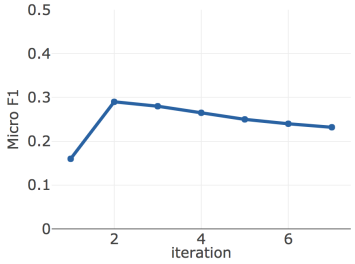
ACM Reference Format:
Aditya Pal and Deepayan Chakrabarti. 2018. Label Propagation with Neural Networks. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3269206.3269322

1 INTRODUCTION

Label Propagation (LP) is a transductive learning method that infers the labels of nodes in a graph, given the labels of a small subset of the nodes [18], [17]. LP does this inference for each node based on the aggregate labels of their neighbors until the labels for all the nodes do not change. The attractive aspect of LP is its simplicity leading to an efficient *bulk synchronous parallel* model [9]. As a result LP has been utilized for solving industry scale problems, such as inferring user authority [10], completing user profiles in social media [4], devising load-balanced user-cache framework [13], discovering machine generated emails [14], etc. However there are two key limitations of LP that inhibits its effectiveness:

(1) **Decreasing Performance:** We notice that in initial iterations, the performance of LP improves. However, it surprisingly starts to decrease afterwards. Figure 1 shows this trend for Flickr dataset. There is some prior theoretical work [1, 8] that suggests that LP's performance degrades for graphs that do not strongly adhere to the principles of local continuity, i.e. neighboring nodes having similar labels. Thus, while one should theoretically run LP until convergence, the "*stopping criterion*" can affect its performance significantly.
(2) **Ineffective Utilization of Node Features:** In most real-world scenarios, the graph nodes have *side information*. Unfortunately, LP does not directly utilize the node features for inferring their labels, rather it uses a kernel function to compute pairwise similarities between the nodes. Thus, the information present in the features gets compressed into scalar edge weights even before LP starts, limiting its ability to leverage the features effectively.

Figure 1: Performance of LP over Flickr dataset.



One way to overcome LP's ineffective utilization of node features is to combine it with a feature-based classifier over the labeled data. This idea is explored by several prior work, such as, LLP [6], EmbedNN [15], etc. A common theme of these methods is to learn a function $g(\cdot)$ over the node features via graph's local continuity based objective and the classifier based objective. We view this as a *tight coupling* of the node features and the graph structure and this makes these prior methods computationally expensive as well as ineffective under noisy scenarios.

In this paper, we propose an extension of LP called LPNN that overcomes the limitations of LP and yet retain LP's simplicity and parallelizability. We achieve this via a *loose* coupling of LP with a classifier objective. The classifier allows the model to utilize the node features directly as well as prevent LP from accumulating the errors while it aggregates labels from the *neighborhoods*. Our main contributions are as follows:

(1) We propose a **joint loss function** that combines LP with a classifier. The classifier, utilizing the node features directly, prevents LP's error accumulation over iterations.
(2) We provide an **inference algorithm** for LPNN that retains parallelizability of LP and its complexity is only linear in the number of edges in the graph. s
(3) We *empirically demonstrate* the effectiveness of LPNN over several popular baselines.

2 NOTATIONS

We consider as input a weighted directed graph with $n = l + u$ nodes, of which the first l nodes are labeled with a binary vector $\mathbf{y}_i \in \{0, 1\}^c$, and nodes $l+1$ to n are unlabeled. The edge weights are indicated by the matrix $W \in \mathbf{R}_+^{n \times n}$, s.t., W_{ij} indicates the weight of edge from i to j. The function $nnz(W)$ indicates the number of non-zero elements in W. Optionally, all nodes have features $\mathbf{x}_{i:1 \le i \le n}$. *Our goal is to infer* $\mathbf{y}_{i:l+1 \le i \le n}$ *for the unlabeled nodes.*

CIKM '18, October 22–26, 2018, Torino, Italy

ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3269322

3 BACKGROUND AND RELATED WORK

LP algorithm was originally proposed by [18]. The original algorithm clamps the labeled nodes to their input labels and propagates their labels to the unlabeled nodes. Clamping ensures a constant push from the labeled nodes ensuring that the class boundaries are pushed through high density data filaments and settle in the low density gaps. One downside of clamping is that if the initial labels are noisy then the model would perform poorly.

We consider a noise-robust variant of LP [17] as our starting point. It estimates a label distribution matrix $\mathbf{f}$ whose columns $\mathbf{f}_i \in \mathbf{R}^c$ represent the label distribution for each node i. The matrix $\mathbf{f}$ is picked to minimize

$$Tr\{\mathbf{f}^T(I_n - S)\mathbf{f}\} + \mu \sum_{i=1}^{l} \|\mathbf{f}_i - \mathbf{y}_i\|_2^2 + \mu \sum_{i=l+1}^{n} \|\mathbf{f}_i\|_2^2, \qquad (1)$$

where I_n is $n \times n$ identity matrix, $S = D^{-1/2}WD^{-1/2}$ is the normalized graph Laplacian with $D_{ii} = \sum_j W_{ij}$ being a diagonal matrix of node degrees. The first term fits a smooth $\mathbf{f}$ over all nodes, the second term encourages consistency with known labels and the last term is a regularizer.

Another key component of our algorithm is combining the classification objective along with LP. Collective classification using both node features and the links connecting the nodes has been widely explored in genral [5], [7]. A typical approach is to infer a function g over the node features $\mathbf{x}_i$ that minimizes

$$\sum_{i,j=1}^{n} \mathcal{L}_G(g(\mathbf{x}_i), g(\mathbf{x}_j), W_{ij}) + \sum_{i=1}^{l} \mathcal{L}_L(\mathbf{y}_i, g(\mathbf{x}_i)).$$

This combines a graph-based loss ($\mathcal{L}_G$), as in LP, with the loss of a node-specific classifier ($\mathcal{L}_L$). The functional form of $g(\mathbf{x}_i)$ leads to different methods: logistic regression in LLP [6], SVM with Hinge Loss in LapSVM [2], neural network in EmbedNN [15].

Recent work on this problem has focused on representing a node via latent embeddings to infer their labels [3], [16]. The following loss function captures the essence of these models.

$$\sum_{i,j=1}^{n} \mathcal{L}_G(h(\mathbf{x}_i), h(\mathbf{x}_j), W_{ij}) + \sum_{i=1}^{l} \mathcal{L}_L\left(\mathbf{y}_i, g(h(\mathbf{x}_i))\right),$$

where $h(\mathbf{x})$ can be viewed as semi-supervised node embedding.

One drawback of these approaches is that in a noisy settings, the noise from the graph and the features can get amplified due to the tight coupling of the two sources as $\mathbf{x}$ are subject to both the classifier objective as well as the graph objective. Our proposed LPNN model complements these approaches by performing a joint optimization of the LP objective and a classifier-based loss.

4 OUR APPROACH

LPNN model is aimed at satisfying the following three key properties that are not adequately addressed by LP:

(1) *Stability*: The model accuracy should improve and then stabilize after a number of iterations.
(2) *Adaptivity*: The model should adaptively weigh the graph and the features based on their predictive power.
(3) *Flexibility*: For a wider applicability, model should work even when *only* the graph or the graph + features are available.

Stability. An intuitive explanation for the decline in LP's accuracy is that the errors accumulating over the initial iterations overwhelm the propagation of informative labels in the subsequent iterations. This suggests coupling LP's label updates with a *noise reducer*. This noise reducer could be a post-processing classifier that is trained to correct LP's output using the labeled set as training data. However, a better option is to jointly optimize LP and the noise reducer. In general, this idea is encapsulated as follows:

$$\sum_{i,j=1}^{n} \mathcal{L}_G(\mathbf{f}_i, \mathbf{f}_j, W_{ij}) + \sum_{i=1}^{l} \mathcal{L}_L(\mathbf{f}_i, \mathbf{g}_i, \mathbf{y}_i) + \sum_{i=l+1}^{n} \mathcal{L}_U(\mathbf{f}_i, \mathbf{g}_i), \qquad (2)$$

where $\mathbf{f}_i$ is a vector representation for node i and $\mathbf{g}_i$ is the output of the noise reducer for node i. The last two terms reconciles label distribution $\mathbf{f}_i$ with the classifier ($\mathbf{g}$) for the labeled and the unlabeled nodes respectively.

Adaptivity. In order for the model to be *adaptive*, it is desirable for $\mathbf{g}$ to pivot on $\mathbf{f}$ and/or $\mathbf{x}$ depending on their predictive power. To achieve this, we set $\mathbf{g}_i = g(\mathbf{f}_i, \mathbf{x}_i)$, where g is a probability distribution over the labels. In contrast, most prior work consider $\mathbf{f}_i = f(\mathbf{x}_i)$ and $\mathbf{g}_i = g(f(\mathbf{x}_i))$, which can under perform if $\mathbf{x}$ is noisy or the noisy graph leads to noisy $\mathbf{f}_i$. On the other hand, *loose coupling* via the choice $\mathbf{g}_i = g(\mathbf{f}_i, \mathbf{x}_i)$ ensures that the noise in either $\mathbf{f}_i$ or $\mathbf{x}_i$ is not amplified.

While any g can be used, we choose a feed-forward neural network where given an input vector z, stage ℓ of the neural network is denoted as $h^\ell(z) = \text{ReLU}(A^\ell h^{\ell-1}(z) + b^\ell)$. Taking the learnings from the Planetoid model [16], we set

$$g_k(\mathbf{f}, \mathbf{x}) = \frac{exp[h^a(\mathbf{f})^T, h^b(\mathbf{x})^T]a_k}{\sum_{k'} exp[h^a(\mathbf{f})^T, h^b(\mathbf{x})^T]a_{k'}},$$

where $[\cdot, \cdot]$ denotes the concatenation of two vectors, $g_k(.)$ is the k-th component of the output of $g(.)$, and a_k, A, b are neural network parameters.

Flexibility. For our model to retain the flexibility of LP, we let $\mathbf{f}_i$'s be label distributions of nodes ($\|\mathbf{f}_i\|_1 = 1, \mathbf{f}_i \geq 0$). This allows the model to default to LP in the worst case of a random/missing $\mathbf{x}$. Most prior models, such as LLP [6], EmbedNN [15], NGM [3]), set $\mathbf{f}_i = f(\mathbf{x}_i)$, which can perform poorly for noisy/missing features.

With the above design choices, LPNN model minimizes the following loss:

$$\mathcal{L}_{LP} + \lambda_L \sum_{i=1}^{l} KL(\mathbf{y}_i \| \mathbf{g}_i) + \lambda_U \sum_{i=l+1}^{n} KL(\mathbf{f}_i \| \mathbf{g}_i), \qquad (3)$$

where the first term is LP loss (Eq. 1) and the next two terms connects the noise reducer with the LP objective. Parameter λ_L, λ_U are the regularization parameter for the noise reducer. It is easy to note that Eq. 3 is a specific instance of Eq. 2.

Matching the Desired Properties. The first KL-divergence term trains $\mathbf{g}$ close to $\mathbf{y}$ on the labeled nodes. The next KL term pulls $\mathbf{f}$ for the unlabeled nodes towards the error-free $\mathbf{g}$. This prevents error accumulation in the LP part of the loss function satisfying our *stability* requirement. Since $\mathbf{g}$ is a function of both node features $\mathbf{x}$ and the label distribution $\mathbf{f}$, it can adaptively combine the two in a way that gives the best prediction for the class labels, matching the *adaptivity* requirement. Finally, when node features are absent,

Algorithm 1 LPNN

Require: Graph W, Features $\mathbf{x}_{1:n}$, Labels $\mathbf{y}_{1:l}$; Regularization parameters $\mu, \lambda_L, \lambda_U$.

1: Estimate $\mathbf{f}$ via LP (Eq. 1)
2: Normalize $\mathbf{f}$ to a probability distribution
3: **repeat**
4: Learn $\mathbf{g}$ via mini-batch gradient step ($\mathbf{f}$ fixed in Eq. 3)
5: Learn $\mathbf{f}$ via Eq. 3 (keep $\mathbf{g}$ fixed)
6: Normalize $\mathbf{f}$ to a probability distribution
7: **until** convergence
8: **return** $\mathbf{g}_{l+1:n}$

Table 1: Dataset Statistics.

Dataset	c	n	$nnz(W)$	has $\mathbf{x}$
Pubmed	3	19,717	44,338	✓
Citeseer	6	3,327	4,732	✓
Cora	7	2,708	5,429	✓
Dblp	31	1,241,210	11,274,954	✓
BlogCatalog	39	10,312	333,983	✗
Flickr	195	80,513	5,899,882	✗

then $\mathbf{g}$ is simply a function of $\mathbf{f}$. Conversely, when the graph is absent, the loss reduces to a form of transductive learning with a prior. This matches the *flexibility* requirement.

Model Training. To train our model, we adopt an alternating minimization technique (See Algorithm 1). Initially, $\mathbf{f}$ is set via LP. After that, *in each iteration*, we first update the noise reducer $\mathbf{g}$ keeping $\mathbf{f}$ fixed. Then, $\mathbf{f}$ is updated while keeping $\mathbf{g}$ fixed. We note here that $\mathbf{f}$ can be updated in parallel similar to LP. Also, $\mathbf{g}$ can be updated on a single machine as its cost is linear in number of nodes.

Convergence. The individual loss is convex in $\mathbf{g}$ or $\mathbf{f}$ but is not jointly convex. Hence the algorithm may not converge in theory; however, it converges in practice to a local minimum.

Time Complexity. The time complexity for computing $\mathbf{f}$ is $O(nnz(W))$, which uses updates similar to LP. The time complexity for learning $\mathbf{g}$ is $O(n)$. For most large datasets, $O(nnz(W))$ dominates $O(n)$, and hence, the bottleneck of our model is the LP step which can be effectively addressed via parallelization.

5 EXPERIMENTS

Datasets. We selected four publicly available academic citation datasets and two social media datasets for our experiments. See Table 1 for statistics of the selected datasets.

LPNN Setup. As in EmbedNN [15], we choose $\mathbf{g}$ to be a 2-layer neural network with 128 and 64 hidden units in the first and the second layer respectively. We use 10% dropout to avoid over-fitting in neural networks. We set LP parameter $\mu = 0.99$ as suggested in [17]. The LPNN parameters λ_L, λ_U are tuned using a 10% hold-out set from the training data.

Baseline Methods

We selected the following baselines.

Table 2: Micro F1 for multi-class classification task using 10-fold cross validation with 10% data per fold for training. The best models are shown in bold if they are statistically significantly better than the second best).

	DNN	EmbedNN	LLP	LP	LPNN
Dblp	0.44	0.81	0.78	0.53	**0.87**
Pubmed	0.43	0.76	0.71	0.63	**0.85**
Citeseer	0.37	0.64	0.58	0.45	**0.68**
Cora	0.66	0.76	0.74	0.68	**0.78**
BlogCatalog	0.11	0.26	0.29	0.28	**0.35**
Flickr	0.14	0.30	0.27	0.26	**0.34**

Table 3: Micro F1 of LPNN for different choices of λ.

	0.1	0.5	1
Dblp	0.81	0.85	0.85
Pubmed	0.85	0.83	0.84
Citeseer	0.63	0.64	0.62
Cora	0.70	0.72	0.74
BlogCatalog	0.34	0.35	0.35
Flickr	0.33	0.34	0.34

LP extensions: We consider the normalized variant of LP [17] and its feature based extension Logistic label propagation (LLP) [6].

We also report the performance of a neural network classifier (DNN) over the node features and also consider a popular transductive classifier EmbedNN [15]. Note, we do not report the results for traditional classifiers such as SVM, Decision Trees, or Logistic regression since they have been shown to under-perform in the transductive setting [15], [16].

5.1 Multi-Class Classification

Table 2 shows the Micro F1 of different methods in a 10-fold cross validation setting with each fold using 10% of the labeled nodes for training and 90% for testing. We note that LPNN performs significantly better than the baselines on all datasets.

In a direct comparison with LP and its extensions (LLP and EmbedNN), we note that LPNN is consistently better across all datasets with a performance gain of 5-30%. We note here that neither the DNN classifier nor LP performs good. However LPNN that combines both LP and DNN not only outperforms these methods but also other state-of-art methods.

Intuitively, the effectiveness of LPNN is due to its coupling of LP with the feature-based classifier. The label distributions $\mathbf{f}$ is effective at encoding local continuity arising from the underlying graph and then $\mathbf{g}$ learns a non-linear decision surface utilizing the information encoded by $\mathbf{f}$ alongside node features $\mathbf{x}$. This loose coupling of the two objectives mitigates the noise coming from either the graph or the features.

5.2 Parameter Sensitivity

LPNN model has two key parameters: λ_L, λ_U that control the label consistency of the labeled data and the unlabeled data, respectively. So far in our experiments, these two parameters are tuned based on

Figure 2: *Classification accuracy against number of iterations:* The accuracy of LP decreases beyond a point, instead of stabilizing at its maximum. LPNN is both more accurate and more stable.

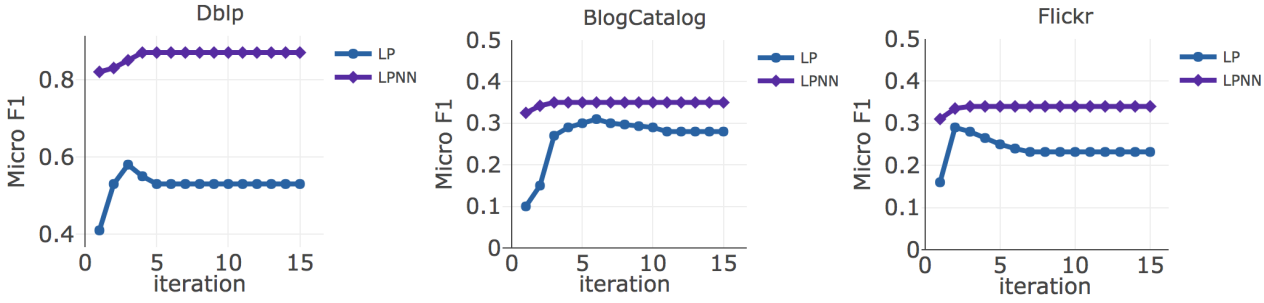


Table 4: Micro-F1 of LPNN variants.

	Dblp	Pubmed	Citeseer	Cora
LP	0.53	0.63	0.45	0.68
NoFeatures	0.59	0.65	0.46	0.71
NoIter	0.80	0.74	0.52	0.73
LPNN	0.87	0.85	0.68	0.78

Table 5: Time taken (in seconds).

	Dblp	Pubmed	Citeseer	Cora
LP	7.9	4.7	1.2	1.1
LPNN	9.4	6.5	3.1	2.9

their performance over a hold-out set. Here we test the sensitivity of the model to the choice of these two parameters. To make the exposition simpler due to space constraints, we set $\lambda_L = \lambda_U = \lambda$ and report the performance of the model for different choices of λ in Table 3. We note that even with the restricted choice of parameters, model performs within a margin of 1-2% w.r.t. to the best performance of LPNN from Table 2.

5.3 Comparison with LPNN Variants

To test the effectiveness of different aspects of LPNN, we consider its two variants: (a) NoFeatures, which runs LPNN without any node features, and (b) NoIter, which runs LPNN for only one round. Table 4 shows that these two variants of LPNN perform better than LP, but are much worse than LPNN. Overall this result justifies our choice of combining $\mathbf{x}$ and $\mathbf{f}$ in the classifier, as well as running LPNN for multiple rounds until convergence.

5.4 Convergence and Time Taken

Figure 2 shows that LPNN converges quickly and its accuracy never decreases, in contrast to LP. We note that LPNN converges quickly and less than 3 rounds of training are required in practice.

Finally, we compare the wall-clock time for our model with LP. We only report the time taken to run the models ignoring the time taken to load the data files in memory. Table 5 shows the time taken on a single processor by the models written in python to converge. Since LPNN runs few iterations of LP only, and it converges much faster, so the overall increase in time is only 10-20%.

6 CONCLUSIONS

In this paper, we proposed the LPNN algorithm to classify objects using both their features and the graph between them. LPNN synthesizes two commonly-held assumptions: that of global patterns, linking object features to class labels in a way that applies globally to all objects, and local continuity, the idea that similar objects tend to have the same label. We showed how LPNN operationalizes these assumptions and avoids the accuracy issues that afflict LP, while remaining computationally feasible, easy to implement, and significantly better than several popular baselines.

REFERENCES

[1] Morteza Alamgir and Ulrike von Luxburg. 2011. Phase transition in the family of p-resistances. In *NIPS*.
[2] M. Belkin, P. Niyogi, and V. Sindhwani. 2006. Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples. *JMLR* (2006).
[3] Thang D. Bui, Sujith Ravi, and Vivek Ramavajjala. 2017. Neural Graph Machines: Learning Neural Networks Using Graphs. *CoRR* abs/1703.04818 (2017).
[4] Deepayan Chakrabarti, Stanislav Funiak, Jonathan Chang, and Sofus A. Macskassy. 2014. Joint Inference of Multiple Label Types in Large Networks. In *ICML*.
[5] Soumen Chakrabarti, Byron Dom, and Piotr Indyk. 1998. Enhanced hypertext categorization using hyperlinks. In *SIGMOD*.
[6] Takumi Kobayashi, Kenji Watanabe, and Nobuyuki Otsu. 2012. Logistic label propagation. *Pattern Recognition Letters* (2012).
[7] Qing Lu and Lise Getoor. 2003. Link-based classification. In *ICML*.
[8] Ulrike V. Luxburg, Agnes Radl, and Matthias Hein. 2010. Getting lost in space: Large sample analysis of the resistance distance. In *NIPS*.
[9] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *SIGMOD*.
[10] Aditya Pal, Amaç Herdagdelen, Sourav Chatterji, Sumit Taank, and Deepayan Chakrabarti. 2016. Discovery of Topical Authorities in Instagram. In *WWW*.
[11] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *KDD*.
[12] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *WSDM*.
[13] Johan Ugander and Lars Backstrom. 2013. Balanced label propagation for partitioning massive graphs. In *WSDM*.
[14] James B. et al. Wendt. 2016. Hierarchical Label Propagation and Discovery for Machine Generated Email. In *WSDM*.
[15] Jason Weston, Frédéric Ratle, and Ronan Collobert. 2008. Deep learning via semi-supervised embedding. In *ICML*.
[16] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *ICML*.
[17] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. 2003. Learning with Local and Global Consistency. In *NIPS*.
[18] Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. 2003. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *ICML*.