Learning the Weights of Label Propagation in Deep Neural Networks

Kojin Oshiba

Abstract

Label propagation (LP) is a classic algorithm for graph-based semi-supervised learning. Typically, the weights given to LP as inputs are assumed to be known, and are often heuristically set based on the data features. We propose a method for discriminatively learning these weights using a deep neural network whose forward pass mimics the steps of the LP algorithm. Empirical result shows the learned weights from our method achieves better performance in predicting the unlabeled data.

1. Introduction

Label propagation (LP) (Xiaojin Zhu, 2005) is a simple iterative algorithm widely used in graph based semi supervised learning. When labels of some nodes are available, but not for others, LP propagates the label information form the labeled nodes to infer the unlabeled nodes based on the weights of the graph.

The weights of the graph used for propagation is often heuristically determined. For example, one way is to use a RBF kernel to set the weights. However, this can be suboptimal as hyperparameters of the kernel are usually set based on past experience. This motivates us to deviate from heuristics and to optimize the weights for the correctness of the label predictions.

The main idea is to design a deep neural network whose forward pass mimics the steps of the LP algorithm. This is done by considering each layer of the network as a single averaging iteration of LP. The loss we consider is based on a leave-k-out approach, where in each example we hide some of the labeled nodes, and try to predict their labels using the other labeled nodes. For generalization, we propose to parametrize the weights as functions of the node features.

We applied our model to a graph constructed from the CORA data, which is a publication graph for machine learning literatures. LP on weights learned using our model showed an increase in accuracy prediction compared to LP on weights set by a RBF kernel.

2. Background

2.1. Problem Setup

Our work develops a method for semi-supervised learning on graphs. Semi-supervised learning is used in situations Where part of the data has labels but the rest of the data doesn't. In such situations, we are interested in inferring the labels of unlabeled data using those of the labeled data. One common way to do so is to construct a graph where each node represents a data point, and weights of the edges document how similar data points are.

More formally,the data relevant to semi supervised learning problems is:

- $y_L = \{y_1, ..., y_l\} \in \{0, 1\}$: the labels of the labeled nodes.
- $y_U = \{y_1, ..., y_u\} \in \{0, 1\}$: the labels of the unlabeled nodes (unobserved).
- $X_L = \{X_1, ..., X_l\} \in \mathrm{R}^D$: the D dimensional feature for labeled nodes.
- $X_U = \{X_1, ..., X_u\} \in \mathrm{R}^D$: the D dimensional feature for unlabeled nodes.
- Let $y = \{y_L, y_U\}$ and $X = \{X_L, X_U\}$

For simplicity, we assume that the labels are binary. This setup and the model are easily extendable to multi-class cases. Here, the goal is to estimate the unlabeled nodes y_U from y_L and X.

2.2. Label Propagation

Label Propagation is an algorithm where, given a graph, weights of the graph, and the labels of the labeled nodes, predicts the labels of the unlabeled nodes. It does so by iteratively updating the estimated labels of the unlabeled nodes by aggregating the label information from the neighboring nodes according to the edge weights. The algorithm is proven to converge. In fact, the estimates of the labels of the unlabeled nodes can be written in a closed form, involving a normalized weight matrix and a vector of labeled nodes. However, since matrix inversions involved in the closed form solution is costly, we would stick to the

iterated approximation to the closed form solution. The pseudocode of the iterative version of Label Propagation is shown in **Algorithm 1**.

2.3. Weight Matrix of Label Propagation

Weight matrix of a graph used in label propagation is unknown most of the time. Hence, if when using LP in practice, we need to estimate the edge weights before running LP. The most commonly used way to estimate these edge weights are by using an RBF kernel:

$$w_{ij} = exp\Big(-\frac{d_{ij}^2}{\sigma^2}\Big) = exp\Big(-\frac{\sum_{d=1}^{D}(x_i^d - x_j^d)^2}{\sigma^2}\Big)$$

As will be explained in depth in later sections, we will parametrize these weights as a function of features X so as to optimize for the label estimation accuracy.

Algorithm 1: Label Propagation (G, y, W);
Input :

- Graph $G = (V, E)$. V is partitioned into unlabeled nodes U and labeled nodes L.
- Labels $y_L \in \{0, 1\}^{|L|}$.
- Edge weights $W \in \mathrm{R}^{|V|\times|V|}$.

Output: $\hat{y}_U \in \{0, 1\}^{|U|}$. Label prediction for unlabeled nodes.
Initialize $\hat{y_U} \leftarrow c$ (some constant)
Row normalize W
for $t = 1, ..., T$ **do**
  for $u = 1, ..., |U|$ **do**
    $\hat{y}_u^t \leftarrow \sum_{j:(j,i)\in E} \hat{w}_{ji}\hat{y}_j^{t-1}$;
    `// averaging updates`
)
  end
end
return $\hat{y}_U$

3. Related Work

Our work develops a method for semi-supervised learning on graphs. Semi-supervised learning is used in situations when part of the data has labels but the rest of the data doesn't. In such situations, we are interested in inferring the labels of unlabeled data using those of the labeled data. One common way to do so is to construct a graph where each node represents a data point, and weights of the edges document how similar data points are.

In graph based semi supervised learning, there are three key types of literatures with respective upsides and downsides. Our key theoretic contribution is to mitigate the three types of methods.

The first type of literature is on estimating the labels of the unlabeled nodes by propagating the label information of the graph. Semi-supervised learning requires smoothness, meaning that data points that are close in feature space are likely to have similar labels. Hence, the labels should be propagated so as to ensure this smoothness. For example, in Label Propagation (Xiaojin Zhu, 2005), the labels of neighboring nodes in the graph are enforced to be similar based on a quadratic penalty. Optimizing the objective in such label propagation tasks can be interpreted as solving a Laplacian system or as the outcome of a random walk process.

In this respect, there are a couple of variants to LP. For example, (Baluja & Aly, 2008) and (Talukdar, 2009) can be interpreted as LP as a random walk with regularization terms added in the objective. (Zhou & Scholkopf, 2003) can also be thought of as graph smoothing with smoothness constraint and fitting constraints in the objective function. These algorithms can all be interpreted as a Laplacian system as noted above.

The main concern with these types of literatures is that it fails to incorporate the information about the covariates. Since the algorithms propagate the labels, and the covariates are only used in constructing the graph, the covariates of the nodes are not fully integrated to the propagation itself.

On the other hand, the second type of literature is on estimating the labels of unlabeled nodes by propagating the *feature* information (instead of label information). In particular, recent methods have a way of incorporating given graphs instead of constructing the graphs themselves. This has been popular because some modern datasets come with existing networks (as in social networks, publication networks, etc.). One such example is (Mikolov & Dean, 2013) where they apply deep embeddings often used in NLP and other tasks to embed a graph to a low dimensional feature representation. In a situation where only the graph itself is available, (Perozzi & Skiena, 2014) introduced a way to still use the graph embeddings for estimating the labels of the unlabeled nodes. In a more theoretical side of the literature, (Yang & Salakhutdi-nov, 2016) and (Kipf & Welling, 2016) showed that these graph embedding methods have a regularization effect for the loss on the labeled nodes being optimized.

In this type of literature, the loss function being optimized is $\sum_{i\in L} L(y_i, \phi(x_i) + R(\theta, x)$ where the first term is the loss on the labeled nodes and the second term is the regularization term. The goal here is to learn ϕ. For unlabeled nodes, predictions are made using this learned function ϕ. The key advantage here is that we are utilizing the features of the both labeled and unlabeled nodes in a supervised fashion. The key downside is that we are propagating the

features but not the labels. More specifically, the focus of the algorithm is in enhancing the estimation of the labeled nodes, but not the unlabeled nodes. We are relying more on the features of the unlabeled nodes but less on the relationship of the unlabeled nodes with other labeled nodes in the graph.

Finally, the last type of literature is on graph construction itself, and not label propagation. Here, researchers have utilized graph Laplace matrices to make the graph smooth, in the sense that some distance metric between neighboring edges is minimized (Kalofolias, 2016). However, these works on graph construction has not been domain specific. Although weights are learned, they are learned based on the smoothing metrics, and not based on the loss we wish to optimize (e.g. the estimation accuracy of the unlabeled nodes).

Our algorithm is novel because first, the weights of graph is not considered given, as in the first and the second type of literatures. Nor, we construct them purely based on the smoothness measure as in the third type of literature. The graph weights are initialized and then updated so as to directly optimize for the predictive accuracy that we ultimately care about. As we will introduce later, by embedding a graph into a neural network, we can optimize for the graph weights initialized using the methods in the third type of literature (e.g. RBF kernel) so as to ensure smoothness AND high performance on semi supervised estimation tasks.

Second, unlike the classic label propagation algorithms where the labels are propagated, and the modern variants of it where the features are propagated, our method propagate both the labels and the features. Labels are propagated through the standard label propagation technique. In addition we can think of weight optimizations as propagating the features according to the leave-k-out loss, which we will introduce in the next section.

4. Model

4.1. Algorithm Architecture

DeepLP is a T layer neural network whose forward propagation mimics the T iterations of LP. Each layer consists of V nodes that correspond to the nodes from the graph. Nodes from the neighboring layers are connected only if the nodes are connected in the original graph.

All layers share the same weight parameters W. To ensure the probabilistic interpretation of the output label, We must constrain W to be positive and normalized. Hence, while each layer is linear in $\tilde{W}$, where $\tilde{W}$ is the normalized W, it is non-linear in the parameters W. This requires careful attention when optimizing over W.

Finally, the inputs and the update rule of each layer is as follows:

- Input layer f^0:

$$f_i^0 = \begin{cases} y_i, & \text{if } i \in L \text{ and not masked} \quad (1) \\ c, & \text{if } i \in U \text{ or masked} \quad (2) \end{cases}$$

- Hidden layer f^t:

$$f_i^t = \begin{cases} y_i, & \text{if } i \in L \text{ and not masked} \quad (3) \\ \sum_{j:(j,i)\in E} \tilde{w}_{ij} f_j^{t-1}, & \text{if } i \in U \text{ or masked} \quad (4) \end{cases}$$

- Output layer f^T: $\hat{y} = f^T$

The full architecture is visualized in **Figure 1**.

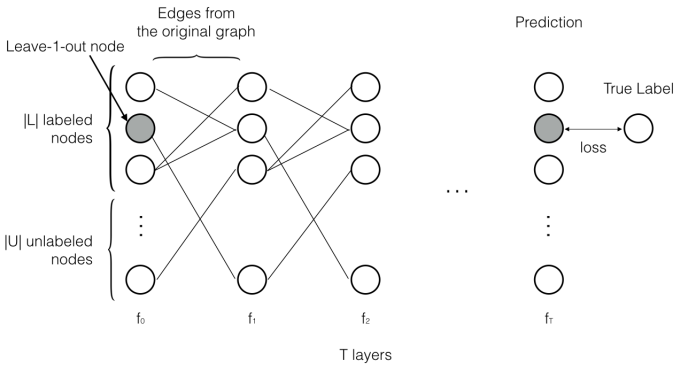

Figure 1. DeepLP Architecture

4.2. Loss Function

We will use a leave-k-out discriminative loss (for simplicity we set k = 1). This means that for each labeled node $i \in L$, we create a single example that is fed into the network. For example, we hide the label of node i, and use the labels of all other nodes (which we denote by y_i) to make a prediction $\hat{y}_i$. $\hat{y}_i$ is compared to y_i to calculate the loss (for a loss function of choice), and the weights are updated according to the back propagation of this leave-1-out loss.

To formalize, let f be the entire DeepLP network, and $\hat{y} = f(y; w)$ be the output prediction. Let $\hat{y}_{leave-i-out}$ or $\hat{y}_{lio}$ be the prediction of the label of the masked label node given other nodes. So, $\hat{y}_{lio} = f(y-i, w)_i, \forall i \in L$. Then, the objective our model is:

$$\min_w \sum_{i \in L} L(y_i, \hat{y}_{lio})$$

4.3. Parameterizing the Weights

The loss given above suffers from an inability to generalize well to unlabeled nodes. This is because edge weights that are not important for propagating label information (such as those not on paths between labeled nodes) are not penalized. The weights we learn for some edges unimportant to propagation can be arbitrary. This motivates us to commonly parametrize all edge weights, so that unimportant edges are tied to other edges with a prior knowledge of how edge weights should be derived from features.

To allow for this generalization, we will use the features $X \in \mathrm{R}^{|L+U| \times D}$:

$$w_{ij} = g(\theta; x_i, x_j)$$

where θ is the new parameter of DeepLP that we want to optimize. The choice of g is a design decision. For example, we can have

- $g(\theta; x_i, x_j) = exp\Big(- \frac{\sum_{d=1}^{D} (x_i^d - x_j^d)^2}{\theta} \Big)$ where θ is a scalar. This is equivalent to optimizing σ^2 in the original model.
- $g(\theta; x_i, x_j) = exp\Big(- \frac{\sum_{d=1}^{D} \theta_d (x_i^d - x_j^d)^2}{\sigma^2} \Big)$ where $\theta \in \mathrm{R}^d$. This is equivalent to making the distance weighted average of each feature, and optimizing those weights.

Once we have g, the objective becomes,

$$\min_w \sum_{i \in L} L(y_i, \hat{y}_{lio}) = \min_w \sum_{i \in L} L(y_i, f(y{-}i, \theta)_i)$$

5. Inference

DeepLP is trained similarly to standard neural networks. Specifically, we will apply back propagation to embedded edge weights. The leave-k-out loss will be optimized using Adam optimizer, and as noted in the previous section, weights are shared across layers.

In addition, we'll introduce regularization on the parameters when it is likely to overfit. For example, for example, when $g(\theta; x_i, x_j) = exp\Big(- \frac{\sum_{d=1}^{D} \theta_d (x_i^d - x_j^d)^2}{\sigma^2} \Big)$, the number of parameters correspond to the number of features, which can often be large. In this case, we let the L1 regularization term to be: $\lambda |\theta - 1|_1$ where λ is the regularization parameter. We subtract 1 here because the defacto is $\theta = 1$, which corresponds to the unweighted average.

We have two kinds of data: the training datas is the data where each labeled node is masked, the validation data is the data where none of the labeled nodes are masked (it is equivalent to the given data). We generate the training data from the validation data.

Based on these two data, we evaluate the performance of the algorithm using 5 metrics:

1. Leave-k-out loss: The loss we directly optimize in DeepLP.
2. Train unlabeled loss: The loss over unlabeled nodes in the training data.
3. Validation unlabeled loss: The loss over unlabeled nodes in the validation data.
4. Train accuracy: The accuracy on unlabeled nodes in the train data.
5. Validation accuracy: The accuracy on unlabeled nodes in the validation data.

The ultimate metric that we care about is the validation accuracy. We will compare the validation accuracy from DeepLP to the baseline accuracy from LP to evaluate the performance.

6. Methods

We compare DeepLP and LP on Cora dataset. The Cora dataset consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 1433 unique words.

For feature engineering, we applied the following transformation.We made the dataset binary by making "Neural Networks" literatures as one category and everything else as another. We also selected maximum connected component of the graph, which is xxx nodes. This is to avoid the adjacency matrix being a singular matrix. Finally, we convert the graph from directed to undirected.

To explore the effectiveness and the characteristics of our model fully, we will run our model on three types of inputs:

1. The input are just the nodes and their features. The graph is constructed on our own. This mimics the situation where a graph is constructed from a given data (e.g. using RBF kernel) for propagation purpose. The graph weights are initial constructed from RBF kernel and is optimized using our model.
2. The input are the nodes, their features and the edges. The graph is considered given. This mimics the sit-

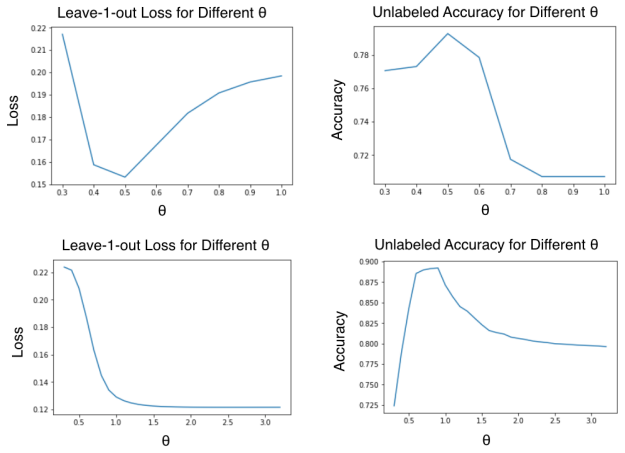

Figure 2. Accuracy, loss, parameter values when a fully connected graph is constructed from the input features (1)

uation where a graph is known or can easily be constructed (as in publication networks, social networks and so on).

In both cases, 95% of the labels are randomly chosen and unlabeled. So our task is to predict these 95% unlabeled nodes from the 5% labeled nodes.

Finally, to see how the choice of g can influence the outcome, we will try two parameterizations we introduced earlier:

- $g(\theta; x_i, x_j) = exp\Big(- \frac{\sum_{d=1}^{D}(x_i^d - x_j^d)^2}{\theta} \Big)$ where θ is a scalar. For convenience, we will call this the scalar divisor parametrization.
- $g(\theta; x_i, x_j) = exp\Big(- \frac{\sum_{d=1}^{D}\theta_d(x_i^d - x_j^d)^2}{\sigma^2} \Big)$ where $\theta \in \mathrm{R}^d$. For convenience, we will call this the weight vector parametrization.

As noted in the previous section, we run both LP and DeepLP to Cora datasets and compare the accuracy on the unlabeled nodes on the validation dataset.

7. Results

Figure3 and the first row of **Figure2** show the result when the graph is constructed from the features and is fully connected (1). The first row of **Figure2** shows the relationship between our leave 1 out loss vs the unlabeled accuracy for different θ in the scalar divisor parametrization. We can clearly see the correlation: when the leave 1 out loss is low, the unlabeled accuracy is high. Hence, we could see that our model which directly optimizes the leave 1 out loss also indirectly optimizes the unlabeled accuracy. The first row of **Figure3** shows how the model was trained every iteration. In the left most plot, we see that the parameter θ approaches to the optimal value. From the middle and the right most plots, we see a smooth decrease in loss and an increase in accuracy, as expected from the first row. The baseline, as we've discussed is the accuracy of Label Propagation, and as we can see from the right most plot, our method outperforms Label Propagation (shown in the orange line). Finally, the second row of **Figure3** is the corresponding plot for the weight vector parametrization. Again, we see that the parameters converge with lower loss and higher accuracy than the baseline.

Figure4 and the first row of **Figure2** show the result when the graph is given (2). In contrast to the case where the graph is constructed using the features, there's no correlation between the loss and the accuracy. Hence, even when we train the scalar divisor, we see that the accuracy doesn't go up (in fact, it goes below the baseline). The reason for the decreasing accuracy is clear from the middle plot in the first row of **Figure4**. For scalar divisor the unlabeled loss goes up when the labeled loss goes down, hence making our leave 1 out loss not a good proxy for the loss on the unlabeled nodes. This is similar in the case of weight vector. In this case, both the labeled and the unlabeled loss go down, but they don't go down enough to make the accuracy go up as well. In fact, the accuracy goes down. Hence, in this case, we can claim that the labeled loss is a good proxy for the unlabeled loss, but not for the unlabeled accuracy.

8. Discussion

The result that our method performs well on the graph when constructed fully based on the RBF kernel and not on the graph that is given is consistent with the theoretical justification of our model. Recall that Label Propagation is motivated by the graph constructed by the features using RBF kernels that modeling the feature density. Hence, using the edges only available on the given graph is like reducing the capacity of optimization by throwing away edge weights that can otherwise be optimized using our method.

In LP, we first approximate the density of feature space by building a graph and then use the graph to propagate the labels. Thoretically, it assumes that the labels are smooth over the density of features i.e. if the two samples are close in feature space than it should be close in labels. The given graph might not leave the edges that are high in RBF kernel weights. Instead, we can think of the given graph as an additional source of information that are different from the node covariates. Hence, incorporating it into the kernel weight generation process can deter the optimization.

Then, the natural question becomes: how can we incorporate the given graph to further optimize the graph weights

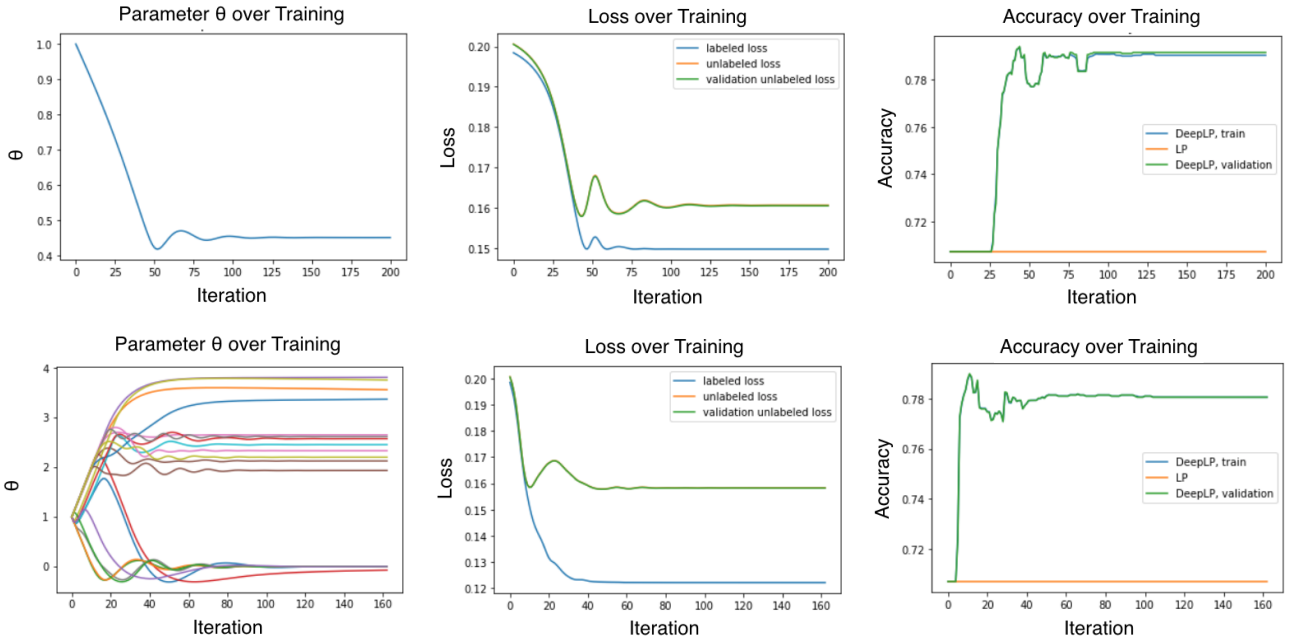

Figure 3. Accuracy, loss, parameter values when a fully connected graph is constructed from the input features (1)

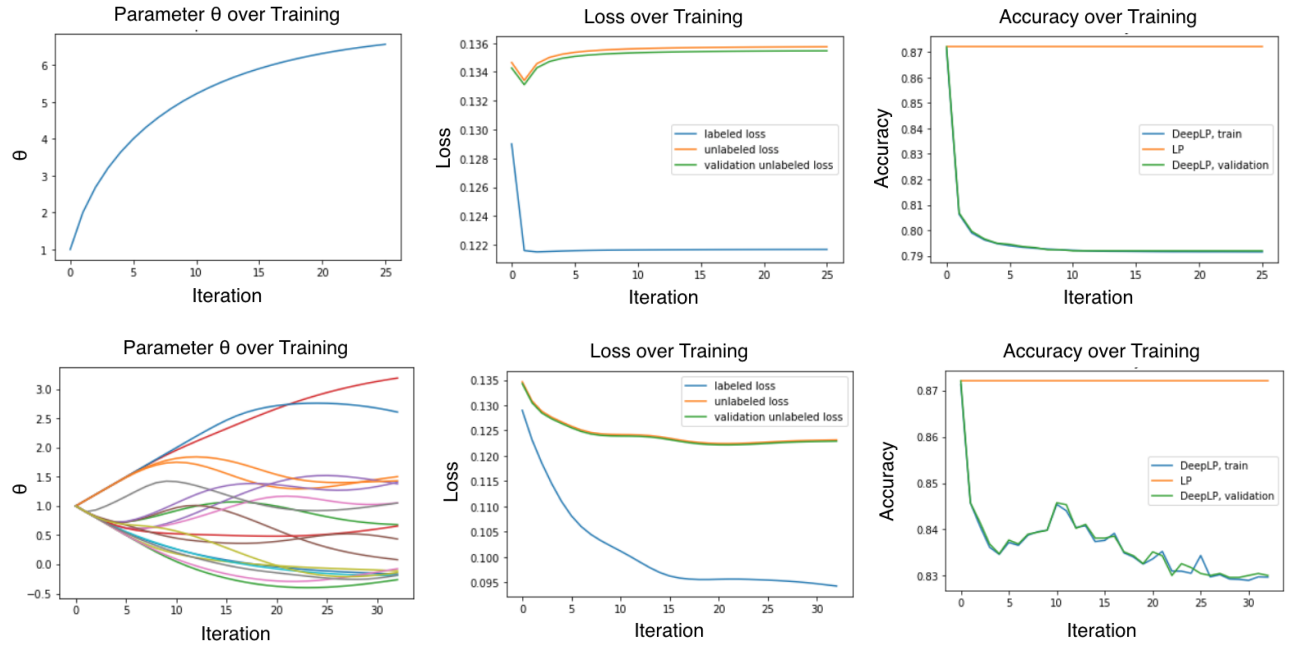

Figure 4. Accuracy, loss, parameter values when a graph is given apart from the features (2)

and edge formation? One approach is to use the given graph as it is and drop the edges that are not in the graph. But as we've shown, the optimization of our model doesn't get along well with such approach. A better approach is to model g not only as a function of covariates, but also as a function of the given graph. Perhaps, we can regularize the weights generated by the kernel (or variants of it as we've exemplified above) by lowering the weights if that edge doesn't exist in the given graph, even if th kernel produces a high weight for that node.

A very different direction we can take is to make g deviate completely from the variants of RBF kernel. This can have several advantages. First, some parametrization of g can not only model similarities but also disimilarities. Since RBF kernel take value 1 which means that the nodes are similar and 0 which means that it is not being considered (as the edge will be dropped), it doesn't encode the information about disimilarity. If we have, for example, g that takes the range of -1 to 1, we can also model disimilarity. Second, we can choose g that are specific to the type of node data. For example, if the nodes are images, then RBF kernel is not the best way to define the similarities of two node images. In such cases, we can perhaps define g as a CNN which can better encode image similarities.

Regardless, parameterizing of graph weights using a function g would grant a great flexibility in the kind of information we can incorporate before learning the labels of the unlabeled nodes. One way to think about Label Propagation is that it is a special case of our method, where g is a RBF kernel. To rephrase this, **our model is the generalization of Label Propagation that allows more flexibility**.

9. Conclusion

Label propagation, since it was proposed in the early 2000s, has been popularized because of its high performance despite its simplicity. We proposed a method that takes this algorithm to allow for more capability. By optimizing the weights for the loss, we not only enabled the weights to be smooth over the features, but also to optimize for the metric we care about i.e. accuracy of the unlabeled nodes. Through the process, we parametrized the weights using the features. In our experiment with two simple parametrizations, this method achieved a great increase in accuracy of the unlabeled nodes. However, we also noticed that the current parametrization doesn't fully take into account the graph given (if there exists any) as it can deter the smoothness of the graph constructed. Future work on the incorporation of these graphs given in our parametrization scheme should bring about further improvements in the graph based semi supervised learning tasks.

References

Baluja, S., Seth-R. Sivakumar D. Jing Y. Yagnik J. Kumar S. Ravichandran D. and Aly, M. Video suggestion and discovery for youtube: taking random walks through the view graph. *WWW*, 2008.

Kalofolias, Vassilis. How to learn a graph from smooth signals. *PMLR*, 2016.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. Technical report, 2016.

Mikolov, T., Sutskever-I. Chen K. Corrado G. S. and Dean, J. Distributed representations of words and phrases and their compositionality. Technical report, 2013.

Perozzi, B., Al-Rfou R. and Skiena, S. Deepwalk: Online learning of social representations. Technical report, 2014.

Talukdar, P. P., Crammer K. New regularized algorithms for transductive learning. Technical report, 2009.

Xiaojin Zhu, Zoubin Ghahramani, John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. *ICML*, 2005.

Yang, Z., Cohen W. W. and Salakhutdi-nov, R. Revisiting semi-supervised learning with graph embeddings. *ICML*, 2016.

Zhou, D., Bousquet O. Lal T. N. Weston J. and Scholkopf, B. Learning with local and global consistency. Technical report, 2003.