Semi-Supervised classification with Graph Convolutional Networks

- Team SMAIL:
- Aman Atman(2020121006)
- Kirthi Vignan(2020122004)
- Rishav Goenka (2019112007)
- Yash Motwani(2020122002)

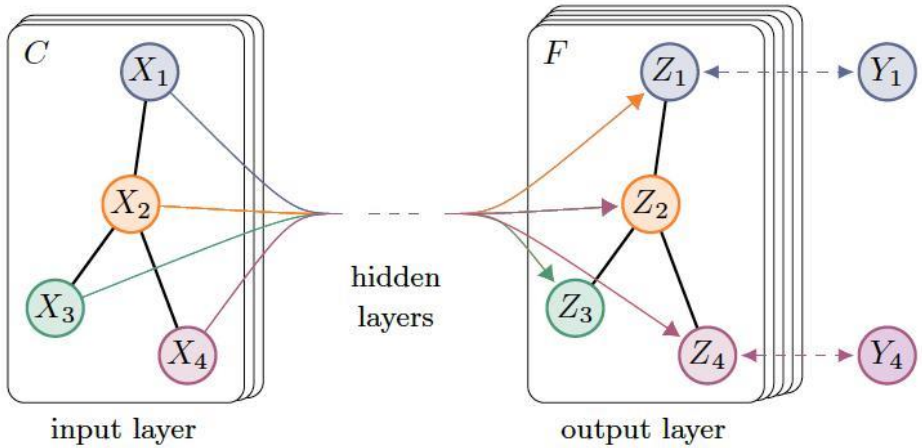
Graph Convolutional Network

The Goal: learn a function of signals/features on a graph $G = (V, E)$

Input:

- A feature description x_i for every node i; summarized in a $N \times D$ feature matrix X (N: number of nodes, D: number of input feature)
- A representative description of the graph structure in matrix form; typically, in the form of an adjacency matrix A (or some function thereof)

Output is a node-level Z (an $N \times F$ feature matrix, where F is the number of output features per node)



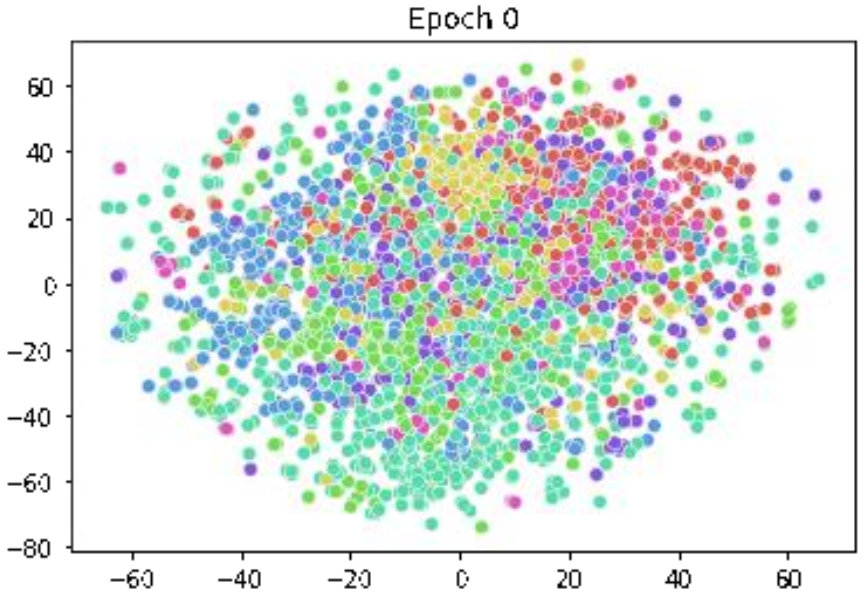
(a) Graph Convolutional Network

Graph Convolutional Network

Every neural network layer can there be written as a non-linear function

$$H^{(l+1)} = f(H^{(l)}, A)$$

with $H^{(0)} = X$ and $H^{(L)} = Z$ (or z for graph-level outputs), L being the number of layers. The specific models then differ only in how $f(\cdot,\cdot)$ is chosen and parameterized.



Cora t-SNE visualization

Loss Function

- Using a graph Laplacian regularization term in the loss function $\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{reg}$, with $\mathcal{L}_{reg} = \sum_{i,j} A_{ij} \left|\left| f(X_i) - f(X_j) \right|\right|^2 = f(X)^T \Delta f(X)$.
- $\mathcal{L}_0$ denotes the supervised loss with respect to the labelled part of the graph
- $f(\cdot)$ can be a neural network-like differentiable function
- λ is a weighing factor
- X is a matrix of node features vectors X_i.
- $\Delta = D - A$ denotes the unnormalized graph Laplacian of an undirected graph $G = (V, E)$ with N nodes $v_i \in V$, edges $(v_i, v_j) \in E$, adjacency matrix $A \in R^{N \times N}$ (binary or weighted) and a degree matrix $D_{ii} = \sum_j A_{ij}$.

Fast Approximate Convolutions on Graphs

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

- $\hat{A} = A + I_N$ is the adjacency matrix of the undirected graph G with added self-connections.
- I_N is the identity matrix
- $D_{ii} = \sum_j \hat{A}_{ij}$
- $W^{(l)}$ is a layer-specific trainable weight matrix
- $H^{(l)} \in R^{N \times D}$ is the matrix of activations in the l^{th} layer
- $H^{(0)} = X$

Spectral Graph Convolutions

$$g_\theta \star x = U g_\theta U^T x,$$

- Filter $g_\theta = diag(\theta)$ parameterized by $\theta \in R^N$
- U is the matrix of eigenvectors of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T$, with a diagonal matrix of its eigenvalues Λ
- $U^T x$ being the graph Fourier transform x.
- However! Evaluating this is computationally expensive, multiplication with the eigenvector matrix U is $O(N^2)$ and computing the eigen decomposition of L in the first place can be expensive for large graphs.

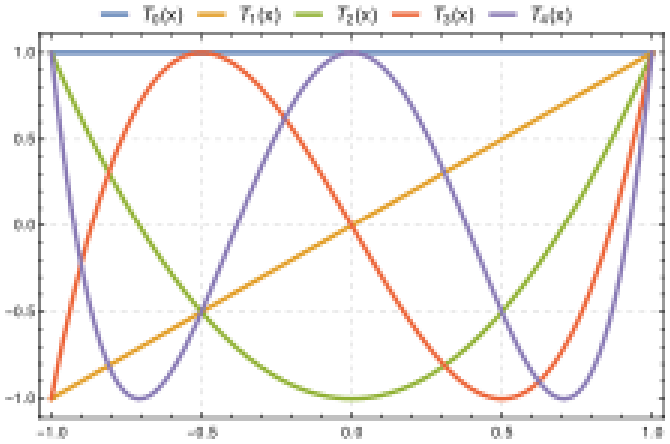
Chebyshev Polynomial Approximation

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\widehat{\Lambda}),$$

Rescaled $\widehat{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I_N$.

λ_{max} denotes the largest eigenvalue of L. $\theta' \in R^K$ is now a vector of Chebyshev coefficients.

The Chebyshev polynomials are recursively defined as
$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$.



Chebyshev Polynomial Approximation

$$g_{\theta'} \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\hat{\mathrm{L}}) x \,,$$

Where $\hat{\mathrm{L}} = \frac{2}{\lambda_{max}} L - I_N$

can be verified by $(U \Lambda U^T)^k = U \Lambda^k U^T$.

Note that this expression is now K-localized since it is a K^{th}-order polynomial in the Laplacian, i.e., it depends only on nodes that are at maximum K steps away from the central node (K^{th}-order neighborhood).

The complexity of evaluating this is $O(|E|)$, i.e., linear in the number of edges.

Stacking multiple convolutional layers of this form, each layer followed by a point-wise non-linearity.

Limit the layer-wise convolution operation to $K = 1$, i.e., a function that is linear with respect to L and therefore a linear function on the graph Laplacian spectrum.

Further Approx.

- Further approximate $\lambda_{max} = 2$, since neural network parameters will adapt to this change in scale during training, the above equation simplifies to

$$g_{\theta'} \star x \approx \theta'_0 x + \theta'_1 (L - I_N) x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x,$$

  with two free parameters θ'_0 and θ'_1. The filter parameters can be shared over the whole graph.
- Successive application of filters of this form then effectively convolve the k^{th}-order neighborhood of a node, where k is the number of successive filtering operations or convolutional layers in the neural network model

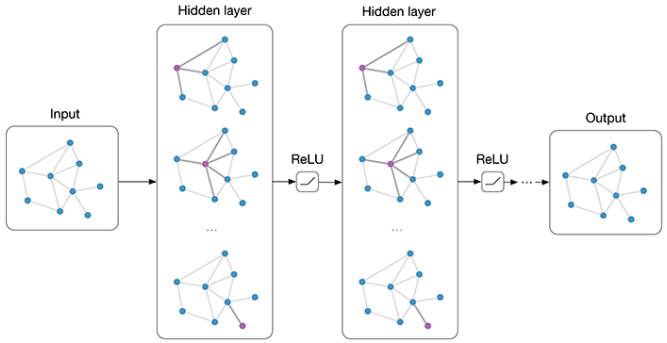
Further Approx.

$$g_\theta \star x \approx \theta \left(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}\right) x,$$

- with a single parameter $\theta = \theta_0' = -\theta_1'$
- $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ now has eigenvalues in the range $[0,2]$ that suggest a renormalization trick, to address numerical instabilities, and/or vanishing/exploding gradients

$$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \widetilde{D}^{-\frac{1}{2}} \tilde{A} \widetilde{D}^{-\frac{1}{2}}$$

$\tilde{A} = A + I_N$ and $\widetilde{D}_{ii} = \sum_j \tilde{A}_{ij}$



Output

- We can generalize the above to a signal $X \in R^{N \times C}$ with C input channels (i.e., a C-dimensional feature vector for every node) and F filters or feature as follows:

$$Z = \widetilde{D}^{-\frac{1}{2}} \tilde{A} \widetilde{D}^{-\frac{1}{2}} X \Theta,$$

where $\Theta \in R^{C \times F}$ is now a matrix of filter parameters and $Z \in R\hat{}(N \times F)$ is the convolved signal matrix. This filtering operation has complexity $O(|E|FC)$

Semi-Supervised Node Classification

- The forward model is then:

$$Z = f(X, A) = softmax\left(\hat{A}ReLU\left(\hat{A}XW^{(0)}\right)W^{(1)}\right)$$

where they first compute in pre-processing:

$$\hat{A} = \widetilde{D}^{-\frac{1}{2}}\tilde{A}\widetilde{D}^{-\frac{1}{2}}$$

Semi-Supervised Node Classification

$$Z = f(X, A) = softmax\left(\hat{A}ReLU\left(\hat{A}XW^{(0)}\right)W^{(1)}\right)$$

Here, $W^{(0)} \in R^{C \times H}$ is an input-to-hidden weight matrix for a hidden layer with H feature maps. $W^{(1)} \in R^{H \times F}$ is a hidden-to-output weight matrix

For semi-supervised multi-class classification, they evaluate the cross-entropy error over all labelled examples as:

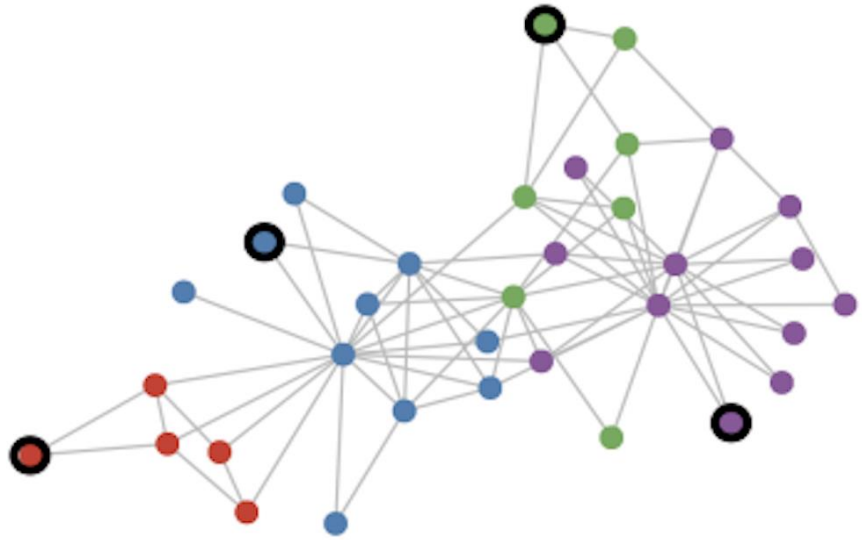
$$\mathcal{L} = -\sum_{l \in \gamma_L} \sum_{f=1}^{F} Y_{lf} ln Z_{lf} ,$$

where γ_L is the set of node indices that have labels.

Semi-Supervised Node Classification for graphs

Setting:
Some node are labeled (black circle)
All other nodes are unlabeled

Task:
Predict nodes label of unlabeled nodes



Evaluate loss on labeled nodes only:

$$\mathcal{L} = -\sum_{l \in \gamma_L} \sum_{f=1}^{F} Y_{lf} ln Z_{lf} ,$$

γ_L set of labeled node indices

Y label matrix

Z GCN output (after softmax)

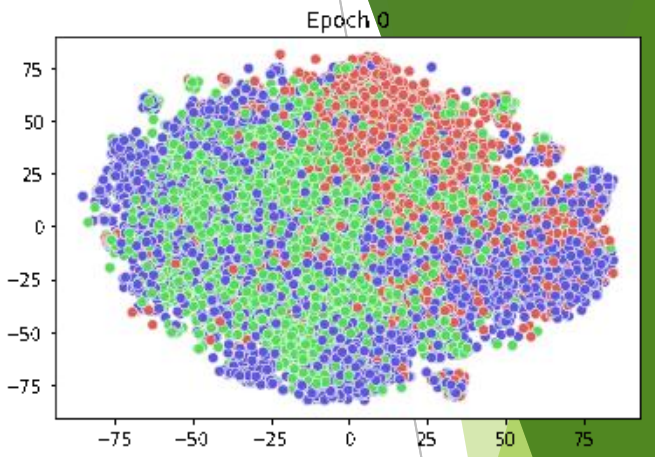
Experimental Set Up

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

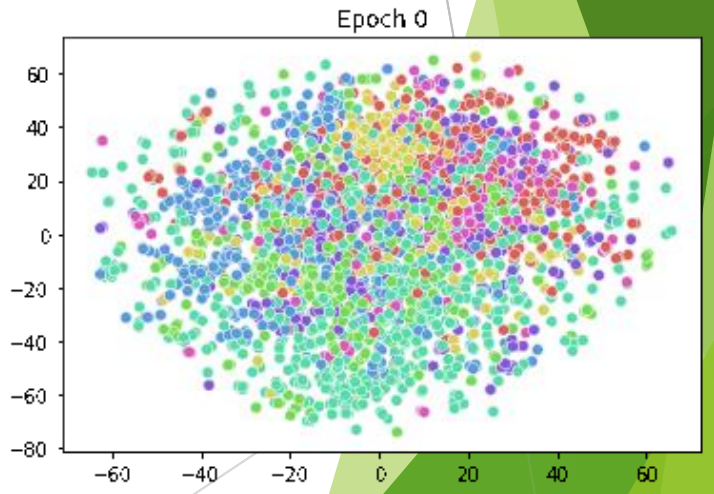
We have used Cora and Pubmed datasets.

Results

Test Accuracy	CORA	PUBMED
200 - iterations	80.30%	78.70%



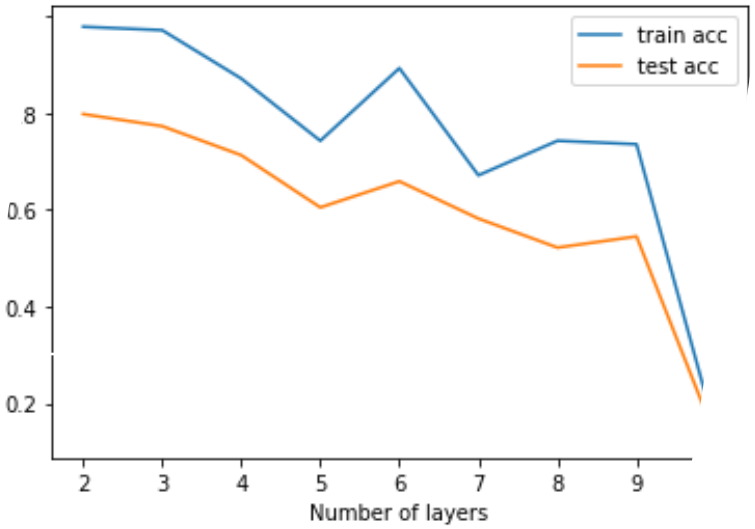
Pubmed t-SNE visualization

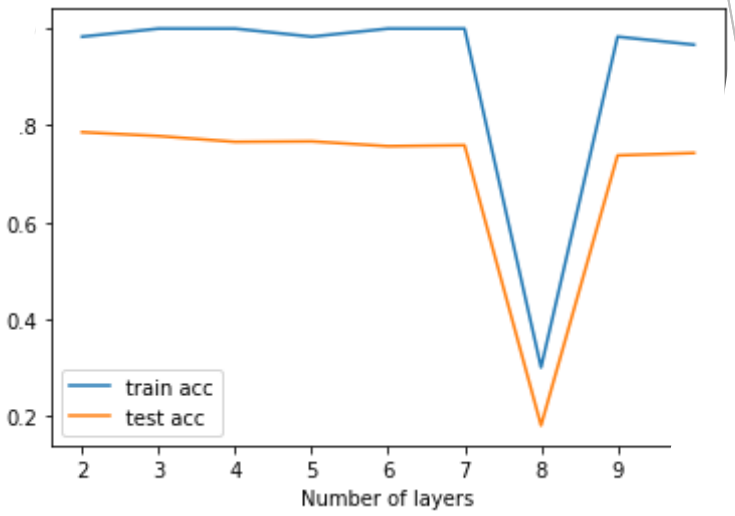


Cora t-SNE visualization

Results

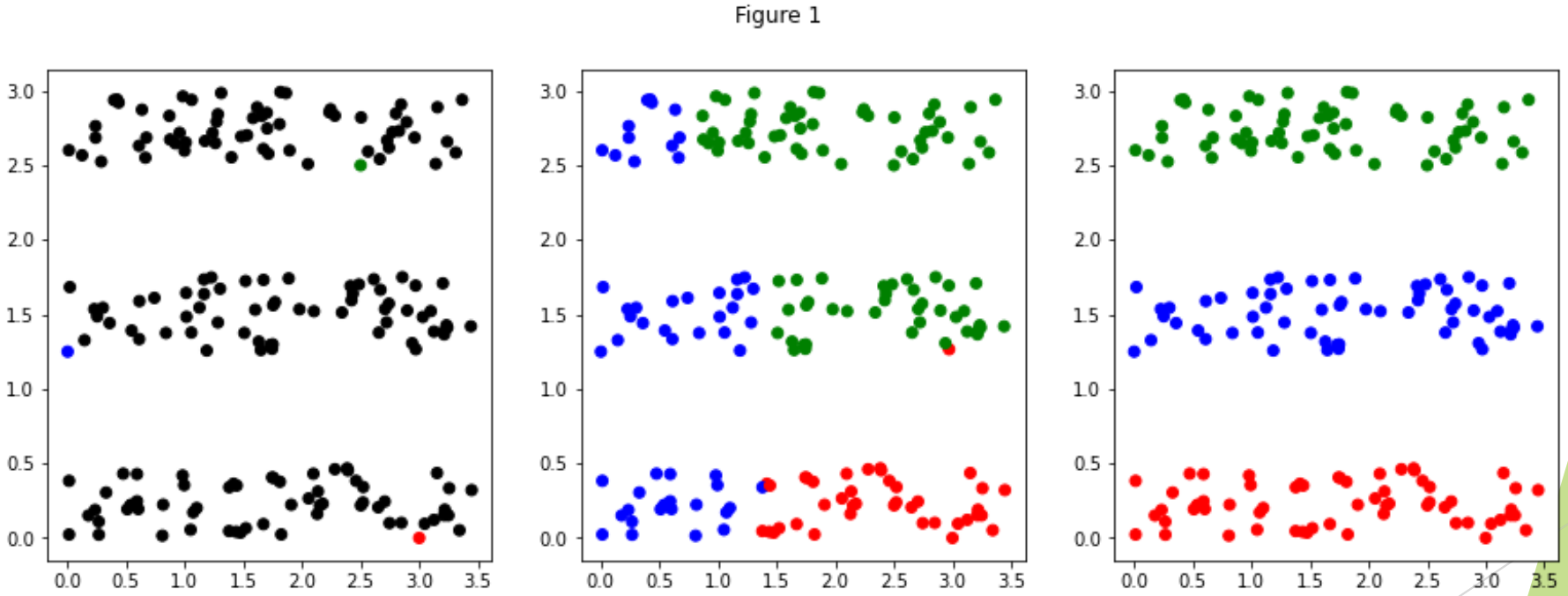
Model depth experiment

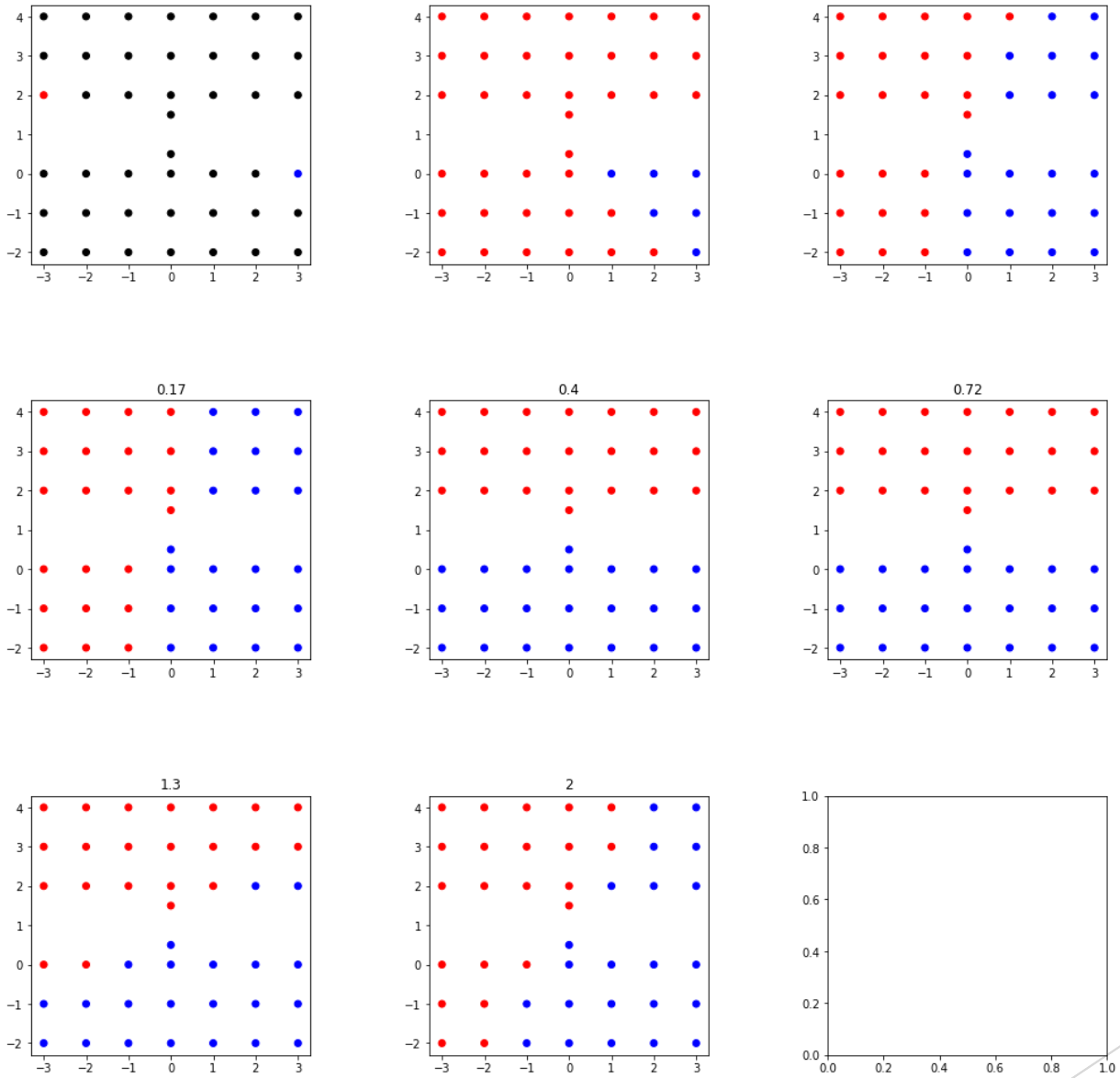




Results

Label Propagation





Label Propagation

```
classifier = Classifier(X[:140], [one_hot[labels[i]] for i in range(140)], 10)
y_lp = classifier.get_pred("lpml", sigma=0.22)
```

```
100 - error_rate(y_lp, labels.tolist())
```

```
99.9996307237814
```

Work Distribution

- Aman Atman and Kirthi Vignan Reddy
  - Writing the code for the label propagation algorithm and GCN from scratch.
  - Implementation of layer-wise linear model based on graph convolution developed.

- Yash Motwani and Rishav Goenka
  - Implementation of spectral graph convolutions.
  - Experiments on model depth i.e., Influence of number of layers on accuracy of proposed model.

Limitations

- Memory grows linearly with data
- Only works with undirected graph
- Assumption of locality
- Assumption of equal importance of self-connections vs. edges to neighboring node

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

where λ is a learnable parameter.