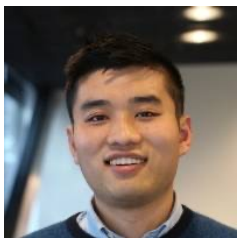


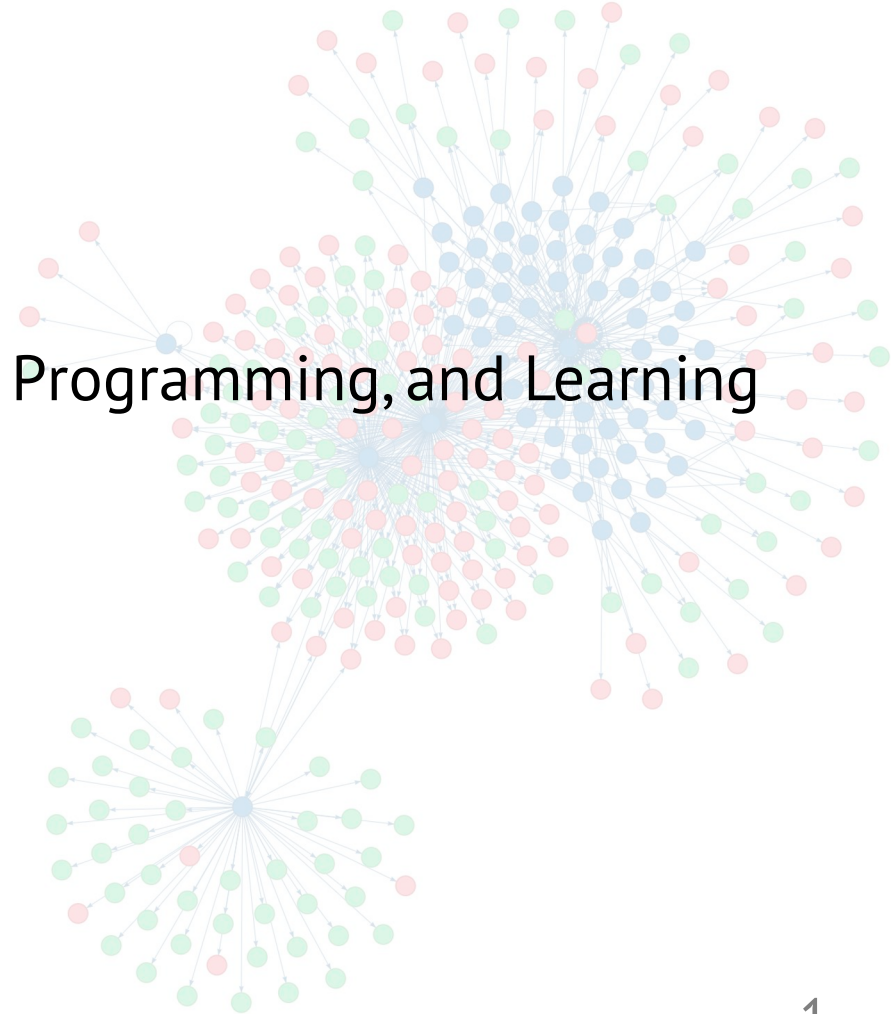
# Label Propagation and Graph Neural Networks

Austin R. Benson · Cornell University

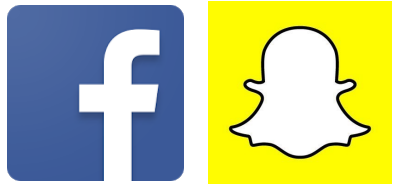
GrAPL 2021: Workshop on Graphs, Architectures, Programming, and Learning



Joint work with  
Junteng Jia (Cornell)



# Graph data modeling complex systems are everywhere.



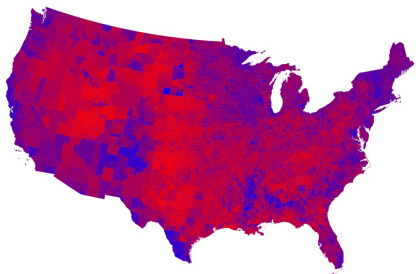
## Society

nodes are people  
edges are friendships



## Finance

nodes are accounts  
edges are transactions



## Elections

nodes are regions  
edges are social / geo

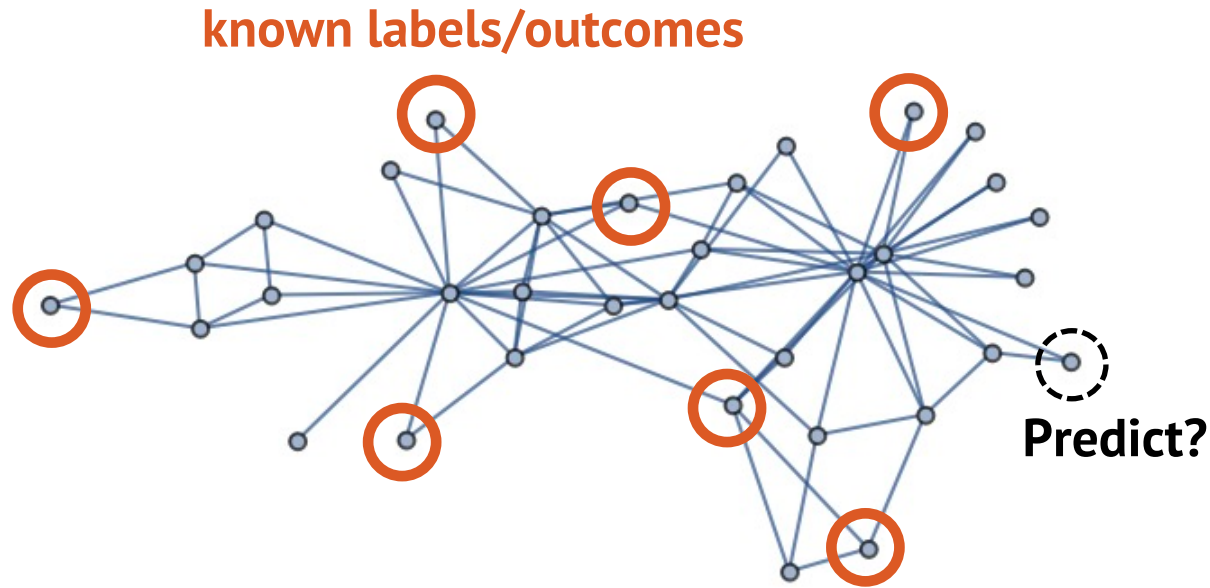
[Mark Newman 2012 map]



## Commerce

nodes are products  
edges are copurchases

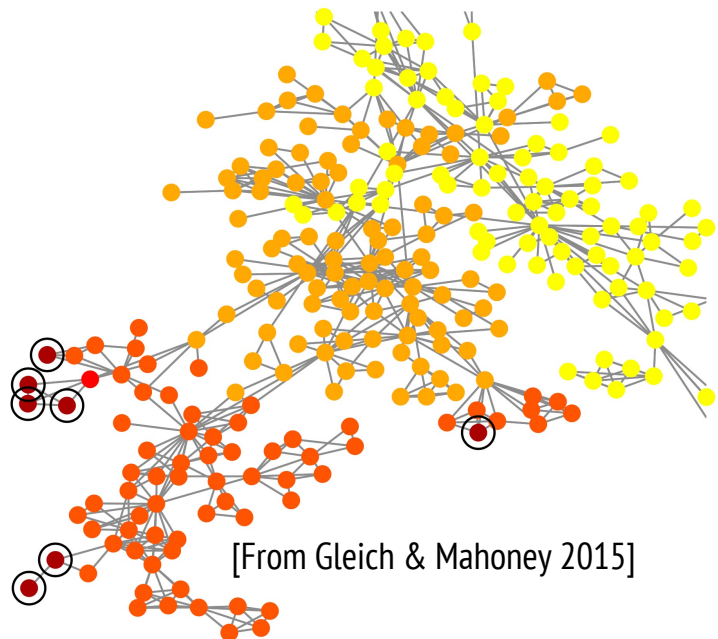
# We often want to predict/estimate/construct/forecast attributes/labels/outcomes/clusters on nodes.



- Bad actors in financial transaction graphs [Weber+ 18, 19; Pareja+ 20]
  - Gender in social networks [Peel 17; Altenburger–Ugander 18]
  - Document classification in citation networks [Lu–Getoor 03; Kipf–Welling 17]
  - Product categories from coreview/copurchase [Huang+ 20; Veldt+ 20]
  - Election outcomes from social connections [Jia–Benson 21]
- 
- Might have rich additional info on nodes (features)  
transaction history, user interests, document text, product ratings, demographics
  - Graph-based semi-supervised learning, clustering, node prediction, relational learning, collective classification, community detection, ...

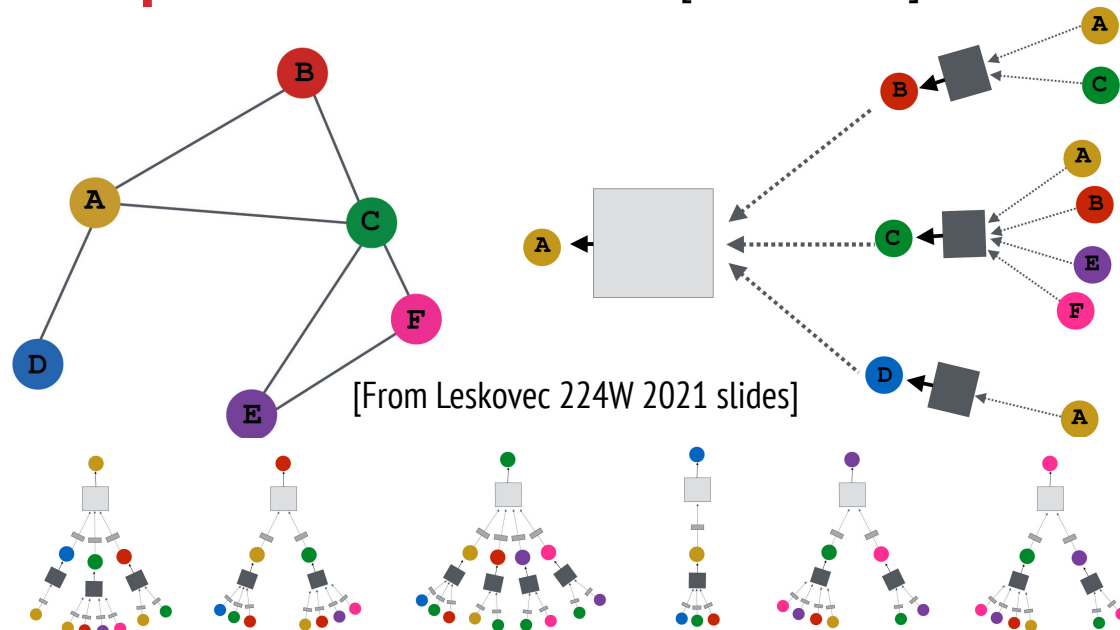
# We will analyze two broad classes of algorithms.

## 1. Label Propagation [early 2000s]



- Propagate/spread/diffuse known values.
- Doesn't use features.

## 2. Graph Neural Networks [late 2010s]



- Combine neighbor features via neural nets.
- Train with known outcomes.
- Produces vector  $h_v$  for each node  $v$ .

### Key questions.

1. When should each work well or poorly?
2. What are the computational tradeoffs?
3. How can we combine them?
4. What is the relationship between them?

# The formal problem we are solving.

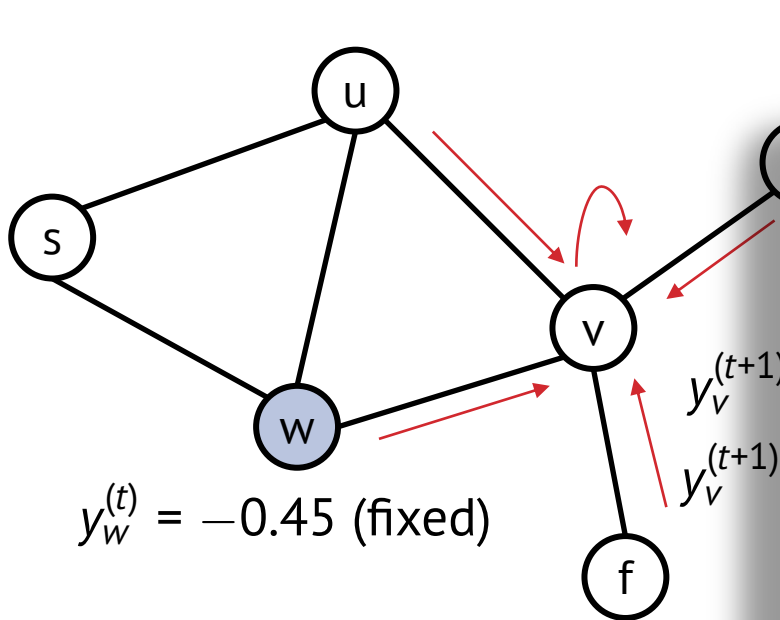
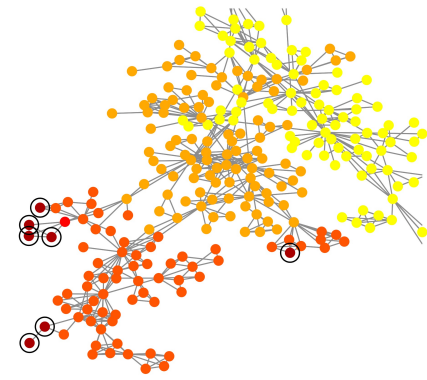
## Problem input.

- Graph  $G = (V, E)$ .
- $|V| \times p$  matrix  $\mathbf{X}$  of node features.
- Subset  $L \subset V$  of labeled nodes.
- Length- $|L|$  vector  $\mathbf{y}_L$  of outcomes on  $L$  (real-valued or categorical).

## Problem output.

- Length- $|U|$  vector  $\mathbf{y}_U$  of real-valued outcomes on  $U = V \setminus L$ .

# Label propagation is just neighbor averaging.



Just need SpMV

$$\mathbf{z} = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2} \mathbf{y}^{(t)}$$

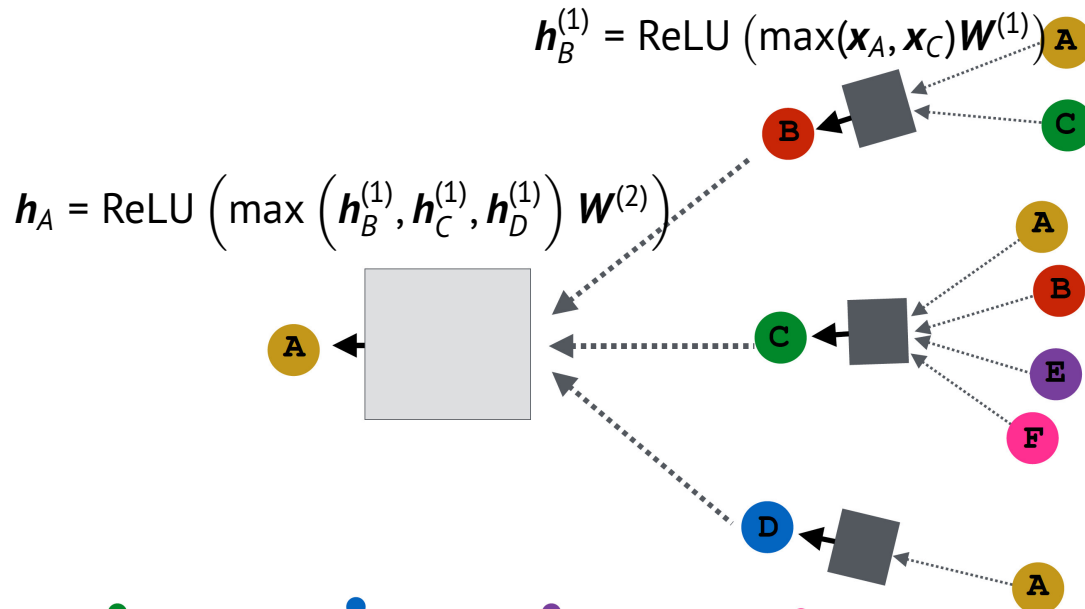
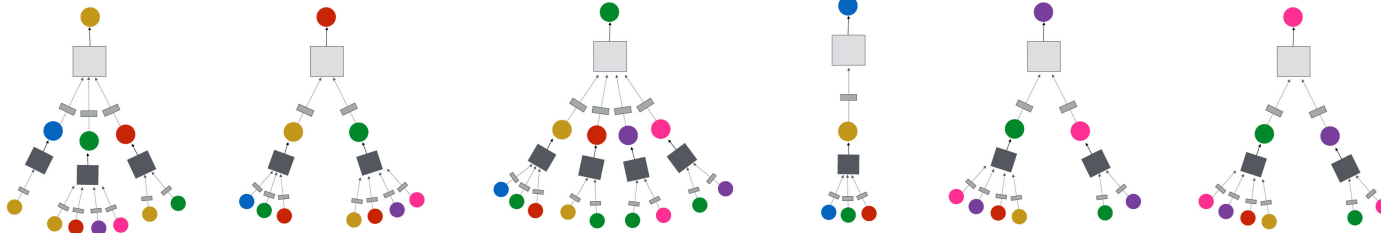
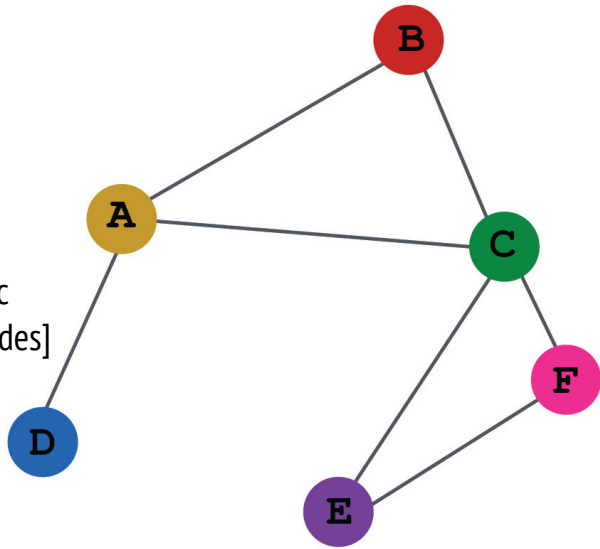
$$\mathbf{y}_U^{(t+1)} = (1 - \alpha) \mathbf{y}_U^{(0)} + \alpha \mathbf{z}_U$$

$$y_f^{(t)} + \frac{1}{\sqrt{3}} y_g^{(t)}$$

- At convergence, everyone is roughly the average over their neighbors → smooth!
- **Regression.** Start with real values (0/mean at unknown) → smoothed value for each node.

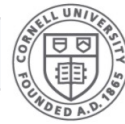
# Graph neural networks aggregate features.

[From Leskovec  
224W 2021 slides]



- **Regression.** Prediction at node A =  $\langle \beta, h_A \rangle$ .
- **BIG** optimization problem trained with labeled nodes and automatic differentiation.
- **DIFFICULT** to implement, parallelize, reproduce.

Chillee commented on Oct 31, 2020 · edited



Hi

I just wanted to verify that the command to reproduce the

(<https://github.com/Espylapiza/dgl/tree/master>)

The comment

# Graph Convolutional Neural Networks for Web-Scale Recommender Systems

Rex Ying<sup>\*†</sup>, Ruining He<sup>\*</sup>, Kaifeng Chen<sup>\*†</sup>, Pong Eksombatchai<sup>\*</sup>,  
William L. Hamilton<sup>†</sup>, Jure Leskovec<sup>\*†</sup>

<sup>\*</sup>Pinterest, <sup>†</sup>Stanford University

{kaifengchen,pong}@pinterest.com, {rexying,wleif,jure}@stanford.edu

# FASTGCN: FAST LEARNING WITH GRAPH CONVOLUTIONAL NETWORKS VIA IMPORTANCE SAMPLING

Jie Chen<sup>\*</sup>, Tengfei Ma<sup>\*</sup>, Cao Xiao

IBM Research

chenjie@us.ibm.com, Tengfei.Ma@ibm.com, cxiao@us.ibm.com

# IMPROVING THE ACCURACY, SCALABILITY, AND PERFORMANCE OF GRAPH NEURAL NETWORKS WITH ROC

Zhihao Jia<sup>1</sup> Sina Lin<sup>2</sup> Mingyu Gao<sup>3</sup> Meta

Chillee commented on

I don't know the details of how you're  
then simply taking the best output from  
seed".

For example, say your only hyperparam  
final loss, but there's also just a lot of n  
hyperparameter tuning might try 0.49,  
the "true" test accuracy much, simply re

# Reducing Communication in Graph Neural Network Training

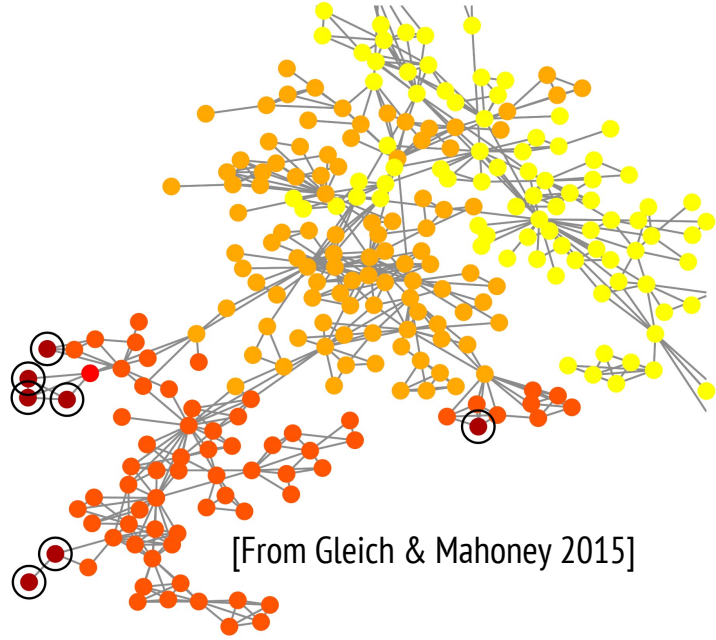
Alok Tripathy, Katherine Yelick, Aydın Buluç

<sup>\*</sup> Electrical Engineering and Computer Sciences, University of California, Berkeley  
<sup>†</sup> Computational Research Division, Lawrence Berkeley National Laboratory



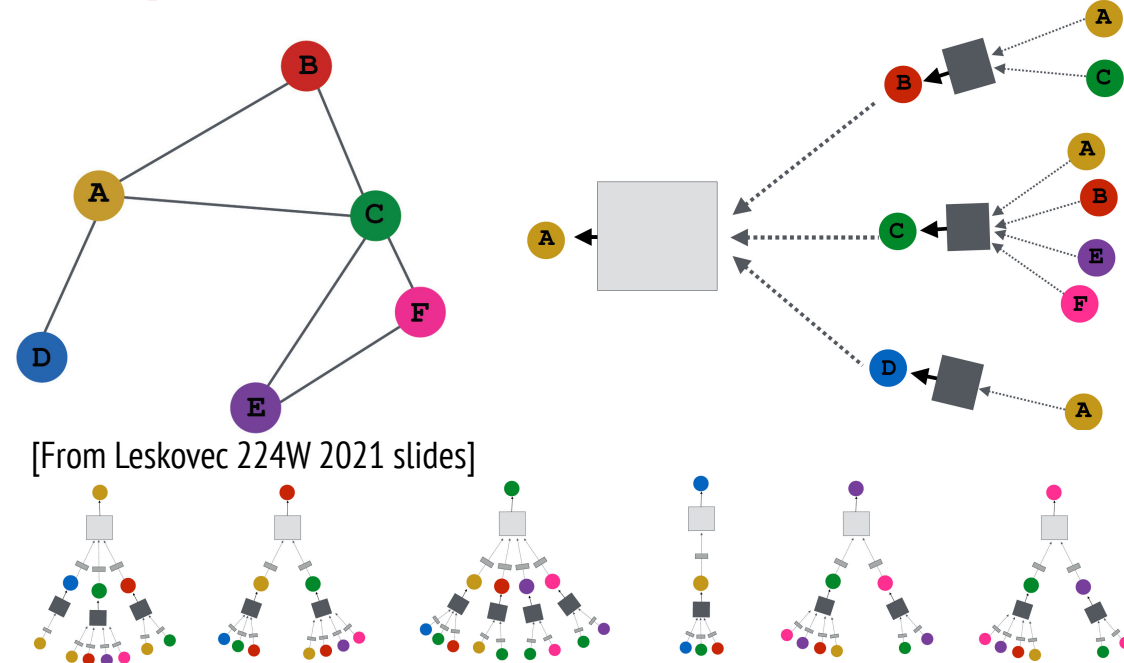
# We will analyze two broad classes of algorithms.

## 1. Label Propagation [early 2000s]



- Strong modeling assumption: connected nodes have similar labels.
- Works because of homophily [McPherson+ 01] a.k.a. assortativity [Newman 02]
- Why not use additional info/features?
- **FAST**  
a few sparse matrix-vector products

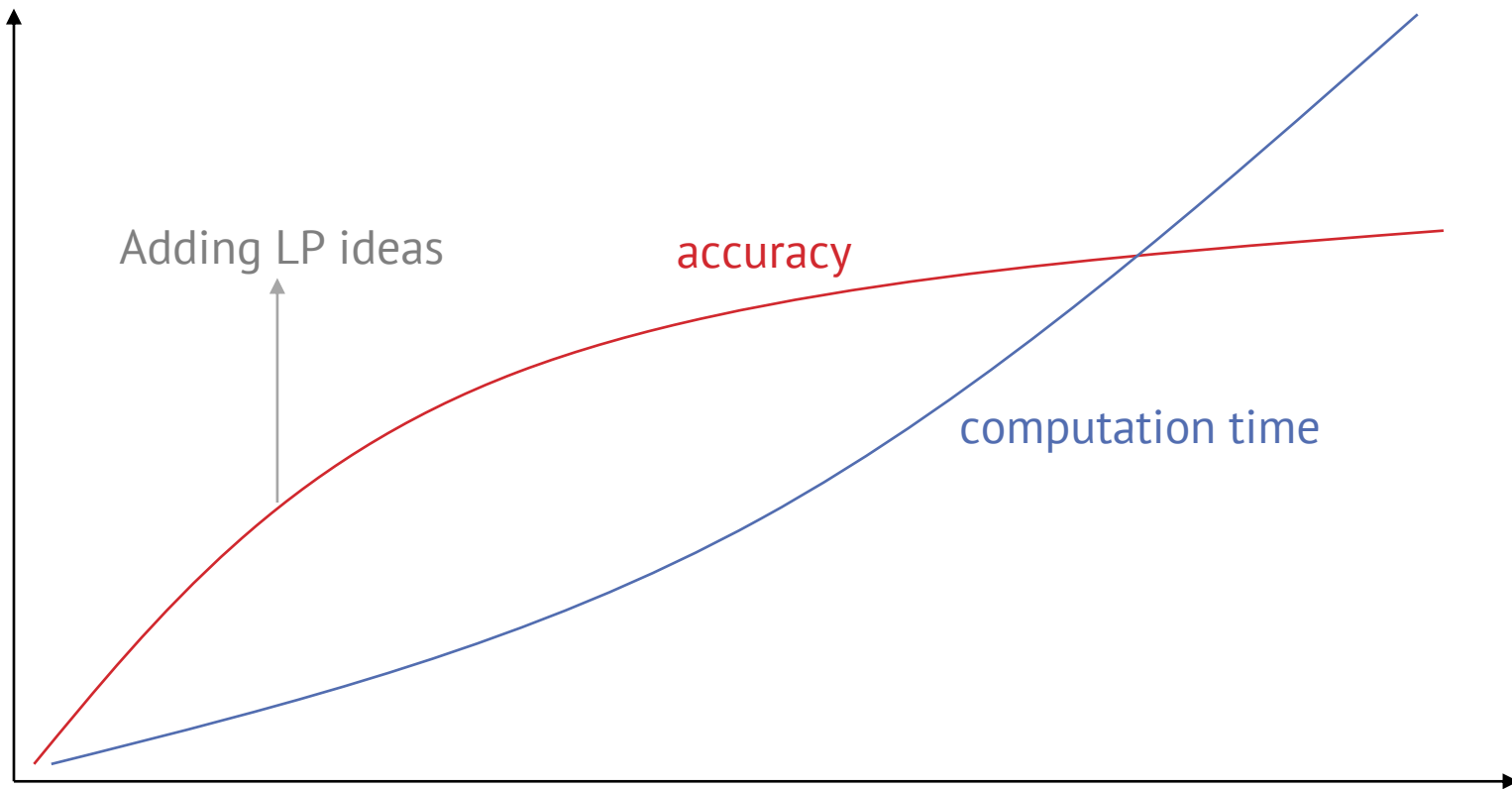
## 2. Graph Neural Networks [late 2010s]



- Strong modeling assumption: labels only depend on neighbor features
- Works because these features are sometimes very informative.
- Why not assume labels are correlated?
- **SLOW**  
many parameters, irregular computation

	Node features	Neighborhood features	Neighborhood labels
Supervised ML (like OLS)	😊		
Label propagation			😊
Graph neural networks	😊	😊	
Our work	😊	😊	😊

Also see *Collective Classification in Network Data* [Sen+ 08] for overview of similar ideas from early 2000s.



More use of node features →  
(bigger & fancier GNNs)

## Leaderboard for [ogbn-products](#)

The classification accuracy on the test and validation sets. The higher, the better.

Package:  $\geq 1.1.1$

Rank	Method	Test Accuracy	Validation Accuracy	Contact	References	#Params	Hardware	Date
1	<b>MLP + C&amp;S</b>	0.8418 ± 0.0007	0.9147 ± 0.0009	<a href="#">Horace He (Cornell)</a>	<a href="#">Paper</a> , <a href="#">Code</a>	96,247	GeForce RTX 2080 (11GB GPU)	Oct 27, 2020
2	<b>Linear + C&amp;S</b>	0.8301 ± 0.0001	0.9134 ± 0.0001	<a href="#">Horace He (Cornell)</a>	<a href="#">Paper</a> , <a href="#">Code</a>	10,763	GeForce RTX 2080 (11GB GPU)	Oct 27, 2020
3	<b>UniMP</b>	0.8256 ± 0.0031	0.9308 ± 0.0017	<a href="#">Yunsheng Shi (PGL team)</a>	<a href="#">Paper</a> , <a href="#">Code</a>	1,475,605	Tesla V100 (32GB)	Sep 8, 2020
4	<b>Plain Linear + C&amp;S</b>	0.8254 ± 0.0003	0.9103 ± 0.0001	<a href="#">Horace He (Cornell)</a>	<a href="#">Paper</a> , <a href="#">Code</a>	4,747	GeForce RTX 2080 (11GB GPU)	Oct 27, 2020
5	<b>DeeperGCN+FLAG</b>	0.8193 ± 0.0031	0.9221 ± 0.0037	<a href="#">Kezhi Kong</a>	<a href="#">Paper</a> , <a href="#">Code</a>	253,743	NVIDIA Tesla V100 (32GB GPU)	Oct 20, 2020



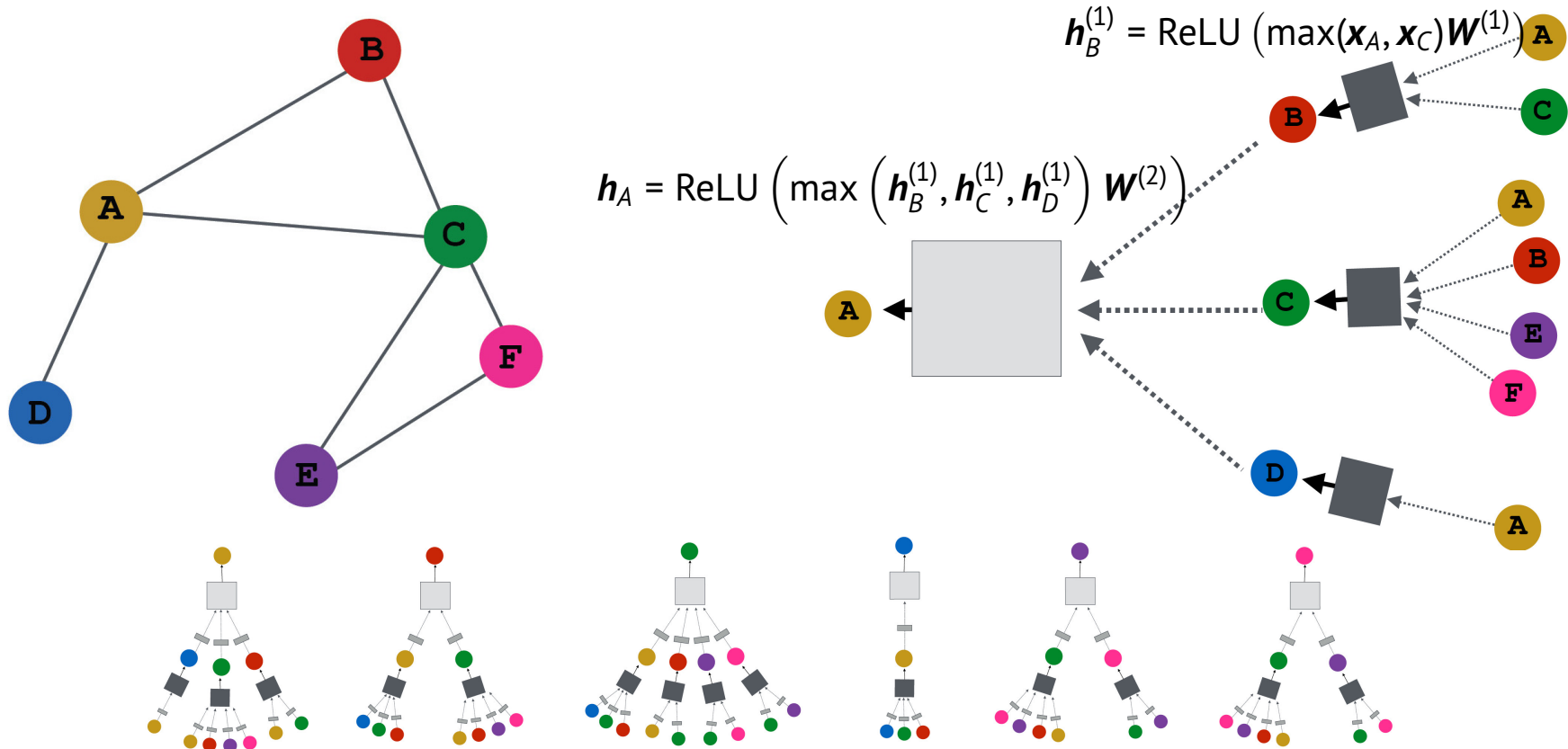
Joint work with  
Qian Huang, Horace He, Abhay Sing (Cornell)  
Ser-Nam Lim (Facebook)

Datasets	Classes	Nodes	Edges	Parameter $\Delta$	Accuracy $\Delta$	Time (s)
Arxiv	40	169,343	1,166,243	-84.90%	+0.37	12 (+90)
Products	47	2,449,029	61,859,140	-93.47%	+1.99	171 (+2959)
Cora	7	2,708	5,429	-98.37%	+1.28	< 1 (+7)
Citeseer	6	3,327	4,732	-89.68%	-0.70	< 1 (+7)
Pubmed	3	19,717	44,338	-96.00%	-0.29	< 1 (+14)
Email	42	1,005	25,571	-97.89%	+4.33	43 (+17)
Rice31	10	4,087	184,828	-99.02%	+1.39	39 (+12)
US County	2	3,234	12,717	-74.56%	+1.77	39 (+12)
wikiCS	10	11,701	216,123	-84.88%	+2.03	7 (+11)

Combining Label Propagation and Simple Models Out-performs Graph Neural Networks.  
Q. Huang et al., ICLR 2021.

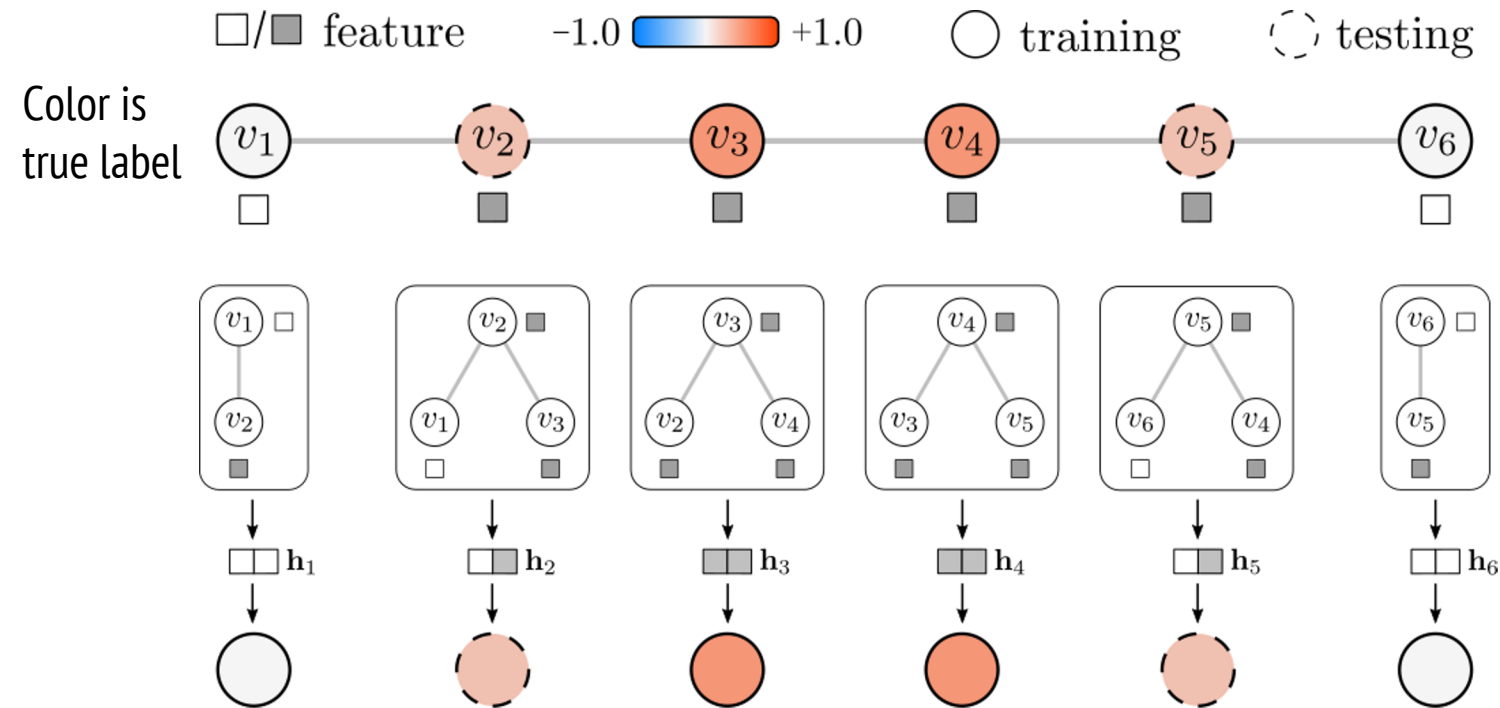
The core problem is that  
(traditional) GNNs  
make uncorrelated predictions.

# Graph neural networks make uncorrelated predictions.



- Use labels to find representation vectors  $h_A, h_B, h_C, h_D, h_E,$  and  $h_F$  and coefficients  $\beta$ .
- Given representations and coefficients, predictions are independent.
- Something strange? Compared to LP, use of labels is very implicit.
- Pervasive paradigm [Kipf-Welling 16; Hamilton+ 17; Zhou+ 18; ~10,000 papers in 5 years]

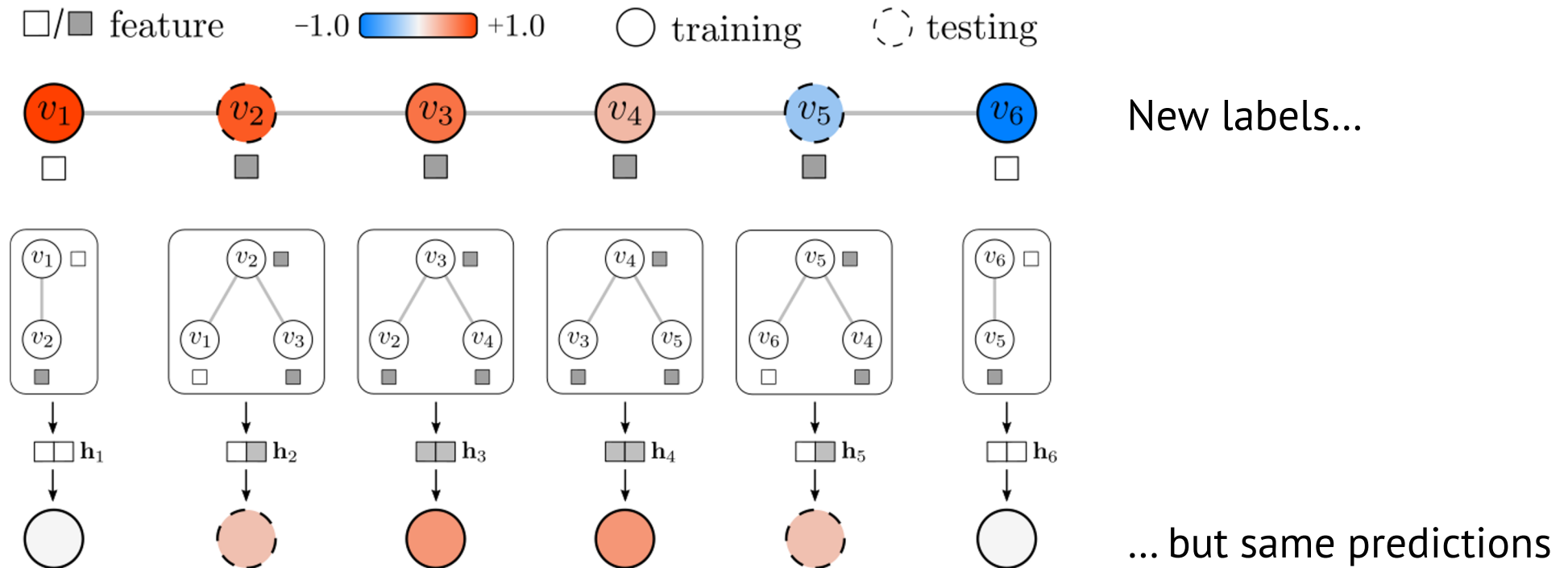
# Graph neural networks make uncorrelated predictions.



- If node features are overwhelmingly predictive, these uncorrelated predictions might be OK.

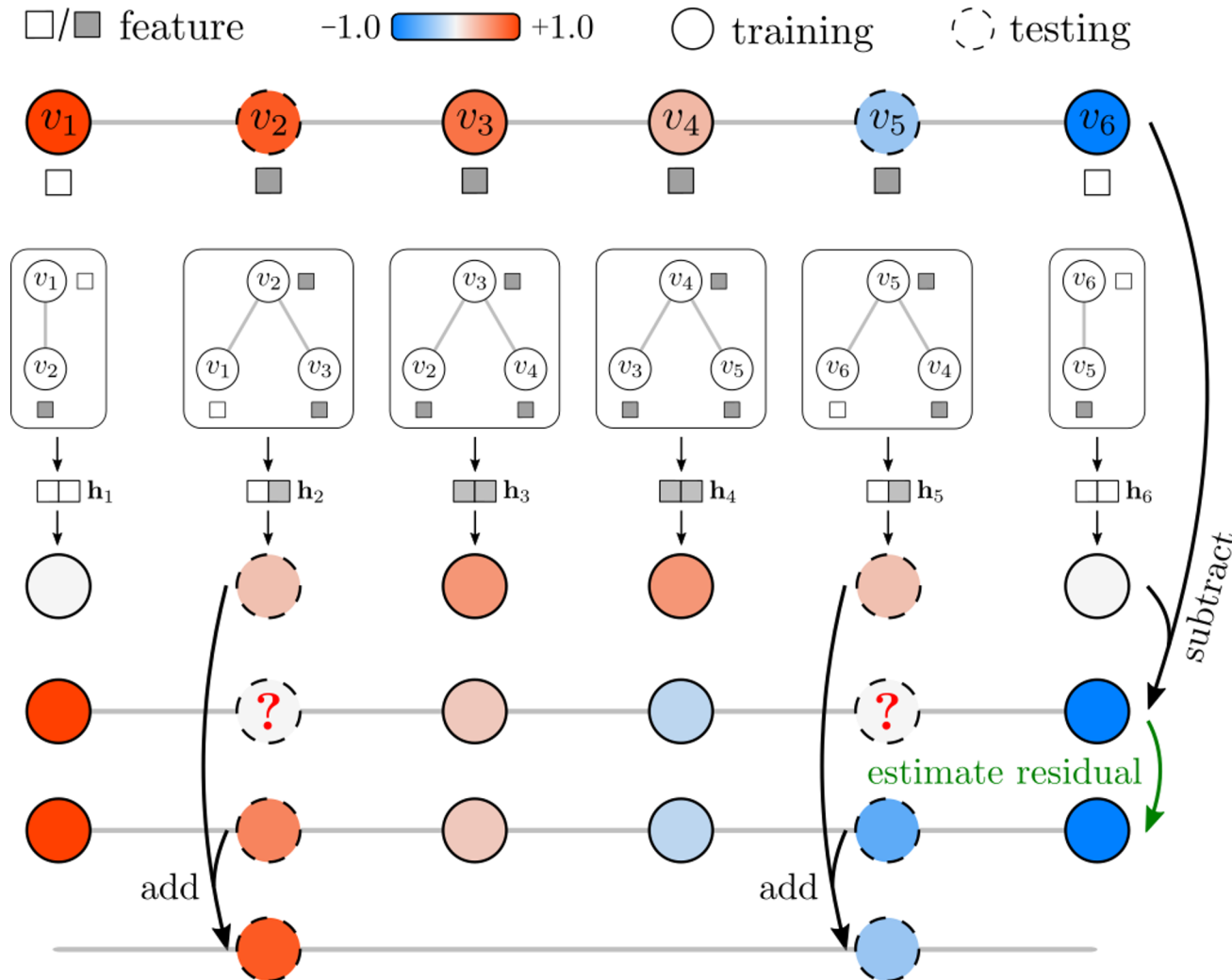


# Uncorrelated GNN predictions can be catastrophic in simple cases when features are only mildly predictive.



- All we have done is change the label distribution!
- **Big problem.** Features are no longer super predictive.
- LP (ignoring features) would work much better.

# We can correlate feature-based predictions by propagating residual errors.



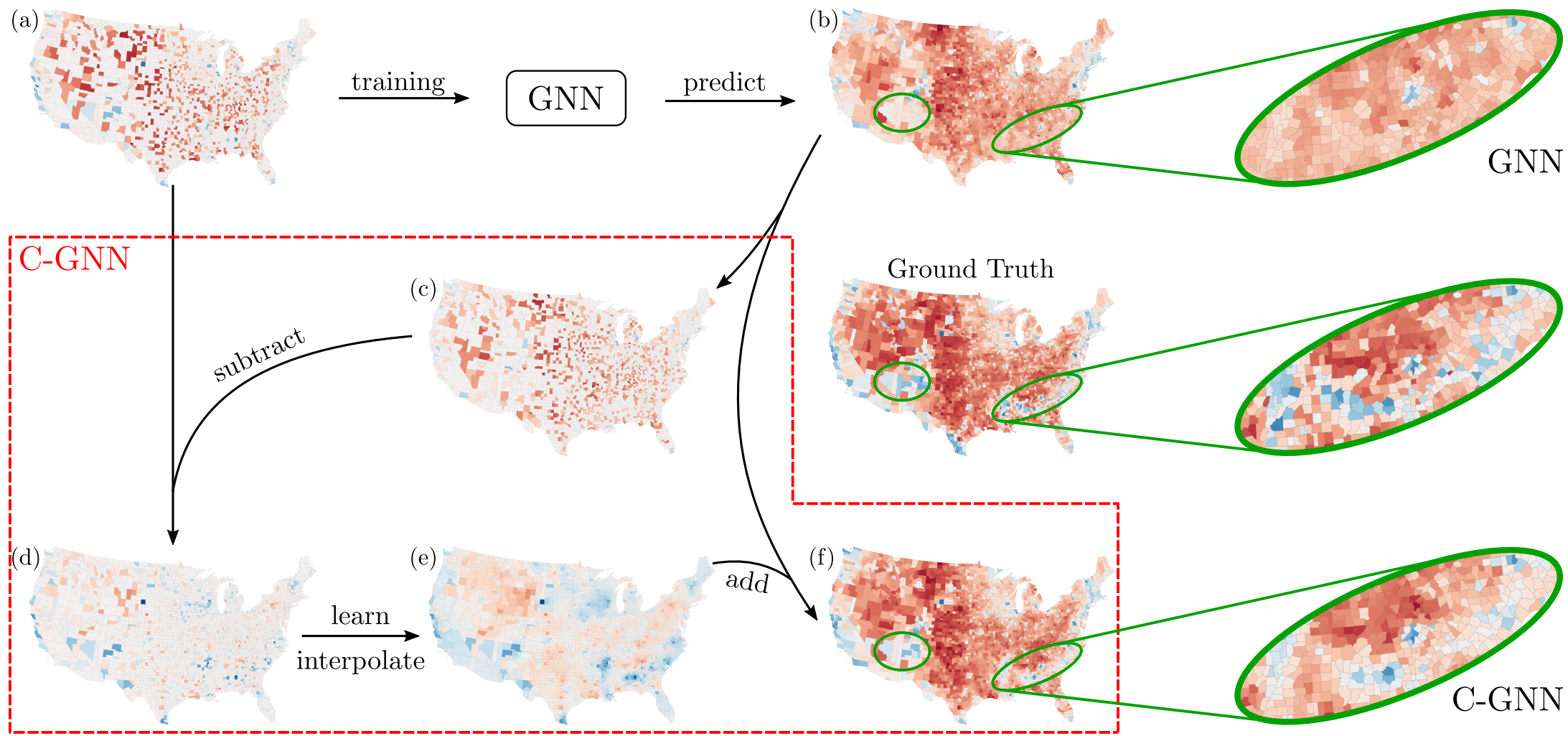
Works with any GNN.  
Just layer on top.

1. Standard GNN prediction.
2. Compute residual error.
3. Propagate residuals to estimate errors on test nodes.
4. Add residual to base prediction.

# The residual propagation algorithm is simple.

1. Make a base prediction on each node with any method.
2.  $\text{residual} = \text{true value} - \text{base prediction}$  (on labeled nodes)
3. Label propagation on residual  $\rightarrow$  smooth errors
4. Final prediction = smoothed residual + base prediction (= true value on labeled nodes)

# Residual propagation works well in practice.



Out-of-sample  $R^2$  0.51  $\rightarrow$  0.69.

# Residual propagation works well in practice.

$R^2$  on county-level demographics predictions.

vote share	0.51 → 0.69
income	0.75 → 0.81
education level	0.70 → 0.72
unemployment level	0.55 → 0.75

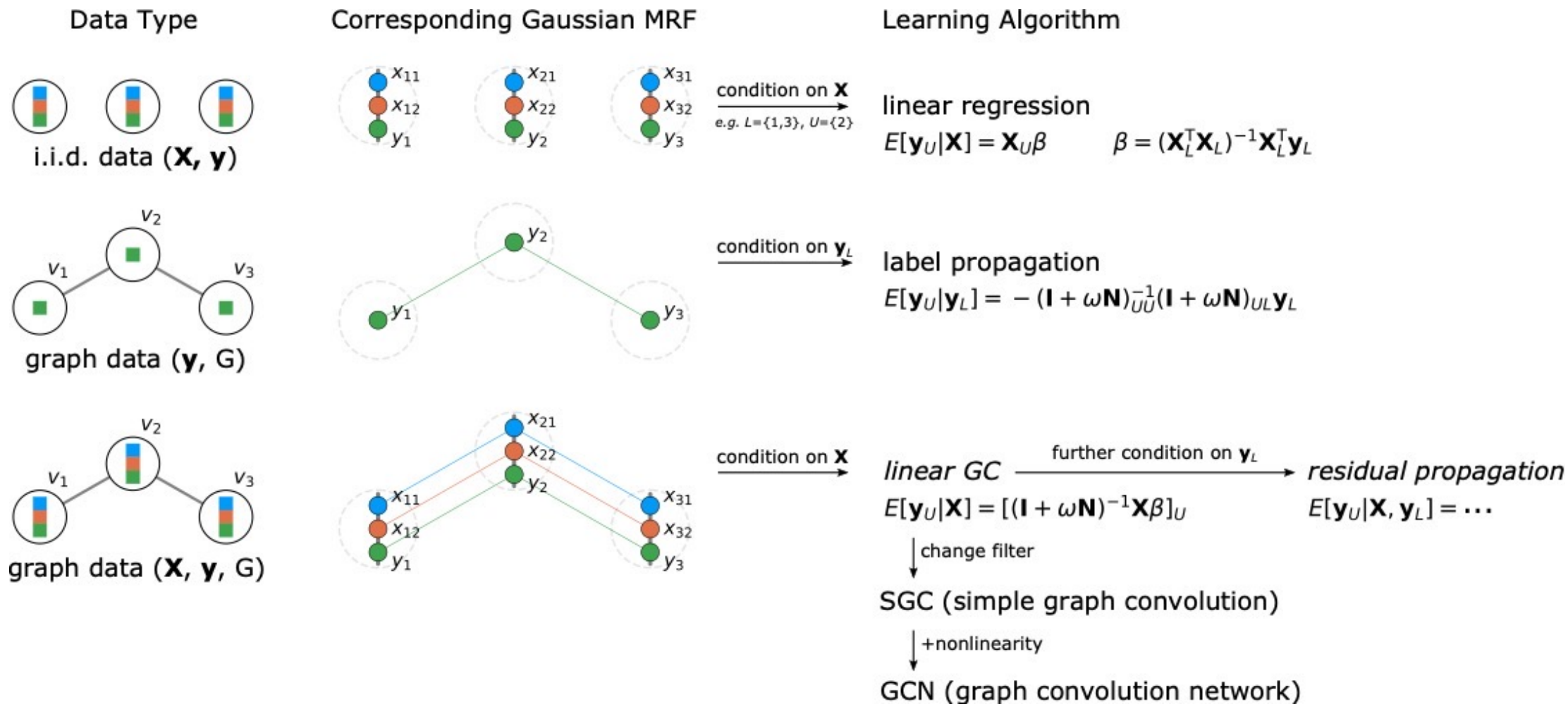
$R^2$  on traffic predictions.

Anaheim	0.76 → 0.81
Chicago	0.68 → 0.72

Why does this work?

Do we need the NN in GNNs?

# We developed a random model for attributes on nodes, where statistical inference leads to GNN/LP algorithms.



# Our model is based on smooth random attributes.

- Random real-valued attribute vectors  $\mathbf{a}_u = [\mathbf{x}_u; y_u]$  on each node  $u$ .
- $\mathbf{A}_i$  =  $i$ th attribute over all nodes.
- $\mathbf{N} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{W} \mathbf{D}^{-1/2}$  is the normalized Laplacian.
- Gaussian MRF random attribute model

$$\phi(\mathbf{A} | \mathbf{H}, \mathbf{h}) = \frac{1}{2} \sum_{u=1}^n \underbrace{\mathbf{a}_u^T \mathbf{H} \mathbf{a}_u}_{\text{correlated attributes on a node}} + \frac{1}{2} \sum_{i=1}^{p+1} \underbrace{h_i \mathbf{A}_i^T \mathbf{N} \mathbf{A}_i}_{\text{smoothness on attributes}}$$

$\mathbf{H} \in \mathbb{R}^{(p+1) \times (p+1)}$  spd,  $\mathbf{0} \leq \mathbf{h} \in \mathbb{R}^{(p+1)}$

$$= \sum_{(u,v) \in E} (A_{ui} / \sqrt{d_u} - A_{vi} / \sqrt{d_v})^2$$

$$\rho(\mathbf{A} = \mathbf{A}' | \mathbf{H}, \mathbf{h}) = \frac{e^{-\phi(\mathbf{A} | \mathbf{H}, \mathbf{h})}}{\int d\mathbf{A}' e^{-\phi(\mathbf{A}' | \mathbf{H}, \mathbf{h})}}$$

Smoother attributes are more likely (homophily / assortativity)

$$\text{vec}(\mathbf{A}) \sim \mathcal{N}(\mathbf{0}, \mathbf{\Gamma}^{-1}), \quad \mathbf{\Gamma} = \mathbf{H} \otimes \mathbf{I}_n + \text{diag}(\mathbf{h}) \otimes \mathbf{N}$$

Just a multivariate normal random variable in the end



# Graph learning is now just statistical inference.

1. Ignore graph, condition on features  $\rightarrow$  linear regression.

$$E[\mathbf{y}|\mathbf{X} = \mathbf{X}] = \mathbf{X}^\top \boldsymbol{\beta} \longrightarrow \min_{\boldsymbol{\beta}} \|\mathbf{X}_L \boldsymbol{\beta} - \mathbf{y}_L\|_2^2 \longrightarrow \mathbf{X}_U \hat{\boldsymbol{\beta}}$$

**(classical derivation of linear models)**

2. Ignore features, condition on graph, labels  $\rightarrow$  label prop.

$$E[\mathbf{y}_U | \mathbf{y}_L = \mathbf{y}_L, \mathbf{G}] = -(\mathbf{I}_n + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I}_n + \omega \mathbf{N})_{UL} \mathbf{y}_L, \quad \omega = h/H$$

**label prop** **Smoothing amount  $\sim$  homophily \* variance**

3. Ignore labels, condition on features + graph  $\rightarrow$  linearized GNN.

$$E[\mathbf{y} | \mathbf{X} = \mathbf{X}, \mathbf{G}] = (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \boldsymbol{\beta} \longrightarrow \min_{\boldsymbol{\beta}} \|[(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X}]_L \boldsymbol{\beta} - \mathbf{y}_L\|_2 \longrightarrow [(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X}]_U \hat{\boldsymbol{\beta}}$$

**label prop**  
**on features**

4. Condition on features + labels + graph  $\rightarrow$  linearized GNN + residual prop.

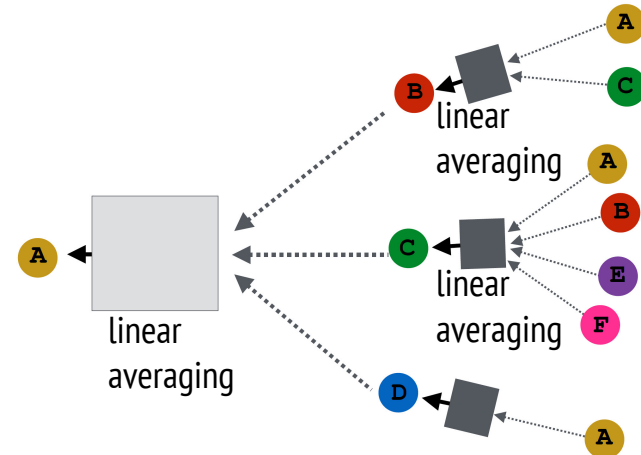
$$E[\mathbf{y}_U | \mathbf{X} = \mathbf{X}, \mathbf{y}_L = \mathbf{y}_L, \mathbf{G}] = \bar{\mathbf{y}}_U + (\mathbf{I} + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I} + \omega \mathbf{N})_{UL} (\bar{\mathbf{y}}_L - \mathbf{y}_L), \quad \bar{\mathbf{y}} = (\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \hat{\boldsymbol{\beta}}$$

**label prop** **residual prop**  
**(on features)**

# Linear graph convolutions are linearized GNNs that come from the conditioning on features.

## Linear graph convolution (LGC).

1. Run LP on each feature  $\rightarrow$  smoothed features.
2. Ordinary least squares on these preprocessed, smoothed features.



# Linear graph convolutions are linearized GNNs that come from the conditioning on features.

**Linear Graph Convolution (LGC)**  $(1 - \alpha) (I + \alpha S + \alpha^2 S^2 + \dots) X \beta$   $S = D^{-1/2} W D^{-1/2}$   
[Jia-Benson 21]

**Simplified Graph Convolution (SGC)**  $\tilde{S}^K X \beta$   $\tilde{S} = (D + I)^{-1/2} (W + I) (D + I)^{-1/2}$   
[Wu+ 19]

**Graph Convolution Network (GCN)**  $\sigma(\tilde{S} \dots \sigma(\tilde{S} X \Theta^{(1)}) \dots \Theta^{(K)}) \beta$   
[Kipf-Welling 17]

- $\alpha$  is continuous, while  $K$  is discrete.
- Does nonlinearity help?
- Does extra parameterization of each “propagation step” in GCN help?
- SGC as  $K \rightarrow \infty$  is nonsensical.

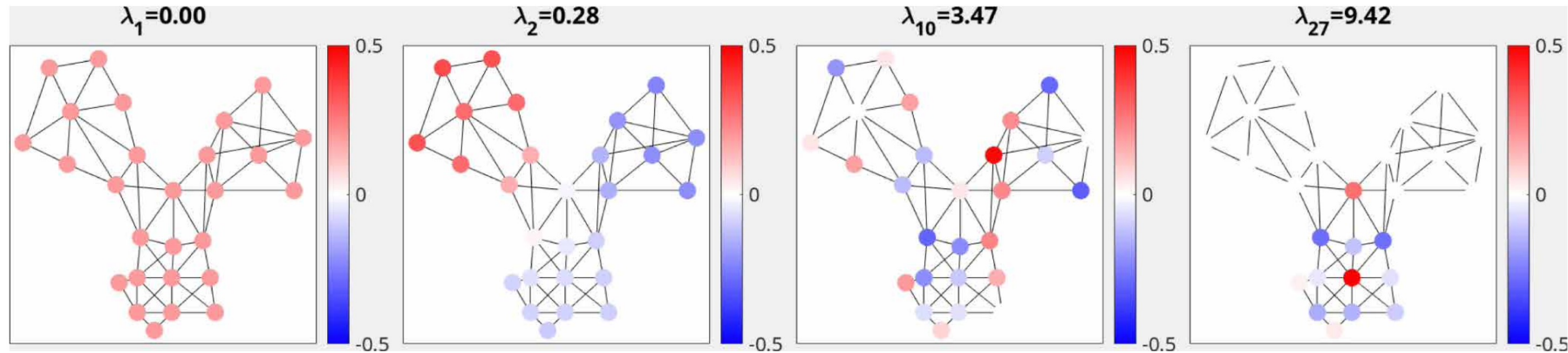
Dataset	Outcome	LP	LR	LGC ( $\alpha$ )	SGC ( $K$ )	GCN ( $K$ )	LGC/RP	SGC/RP	GCN/RP
U.S.	income	0.40	0.63	0.66 (0.46)	0.51 (1.0)	0.53 (1.3)	<b>0.69</b>	0.55	0.55
	education	0.31	<b>0.71</b>	<b>0.71</b> (0.00)	0.43 (1.0)	0.47 (1.0)	<b>0.71</b>	0.46	0.48
	unemployment	0.47	0.34	0.39 (0.59)	0.32 (1.3)	0.45 (2.5)	<b>0.54</b>	0.52	0.53
	election	0.52	0.42	0.49 (0.68)	0.43 (1.1)	0.52 (2.1)	<b>0.64</b>	0.61	0.61
CDC	airT	0.95	0.85	0.86 (0.78)	0.86 (2.6)	0.95 (3.0)	0.96	<b>0.97</b>	<b>0.97</b>
	landT	0.89	0.81	0.81 (0.09)	0.79 (1.0)	0.91 (2.4)	0.90	0.93	<b>0.93</b>
	precipitation	0.89	0.59	0.61 (0.93)	0.61 (2.3)	0.79 (3.0)	0.89	<b>0.90</b>	<b>0.90</b>
	sunlight	0.96	0.75	0.81 (0.97)	0.80 (3.0)	0.90 (3.0)	0.96	<b>0.97</b>	<b>0.97</b>
	pm2.5	0.96	0.21	0.27 (0.99)	0.23 (2.7)	0.78 (3.0)	0.96	0.96	<b>0.97</b>
London	income	0.46	<b>0.85</b>	<b>0.85</b> (0.00)	0.64 (1.0)	0.63 (1.0)	<b>0.85</b>	0.65	0.64
	education	0.65	0.81	0.83 (0.40)	0.74 (1.6)	0.79 (1.4)	<b>0.86</b>	0.77	0.79
	age	0.65	0.73	0.73 (0.17)	0.66 (1.2)	0.70 (1.7)	<b>0.75</b>	0.72	0.72
	election	0.67	0.73	0.81 (0.74)	0.74 (2.0)	0.76 (2.1)	<b>0.85</b>	0.78	0.78
Twitch	days	0.08	0.58	0.59 (0.67)	0.22 (1.4)	0.26 (1.7)	<b>0.60</b>	0.23	0.26

```

1  function LGC_params(S, X, y, L;  $\alpha=0.9$ , num_iters=10)
2      X_smooth = copy(X)
3      for _ in 1:num_iters
4          X_smooth = (1 -  $\alpha$ ) * X +  $\alpha$  * S * X_smooth
5      end
6      return X_smooth, X_smooth[L, :] \ y[L]
7  end
8
9  function residual_prop(S, y,  $\bar{y}$ , U;  $\alpha=0.9$ , num_iters=10)
10     r = y -  $\bar{y}$ 
11     r[U] = 0
12     for _ in 1:num_iters
13         z = S * r
14         r[U] =  $\alpha$  * z[U]
15     end
16     return r
17 end
18
19 function LGC_RP_prediction(
20     S, # normalized adjacency  $D^{-1/2} A D^{-1/2}$ 
21     X, # n x d feature matrix for n nodes
22     U, # indices of unlabeled nodes
23     L # indices of labeled nodes
24     y, # n x 1 label vector (zero on y[U])
25 )
26     X_smooth,  $\hat{\beta}$  = LGC_params(S, X, y, L)
27      $\bar{y}$  = X_smooth *  $\hat{\beta}$ 
28     r = residual_prop(S, y,  $\bar{y}$ , L)
29     return  $\bar{y}$ [U] + r[U]
30 end

```

# Our model helps us understand smoothing.



Graph Signal Processing: Overview, Challenges and Applications, Ortega et al., Proc. IEEE, 2018.

$$\mathbf{N} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T, \text{ feature } \mathbf{f} = \sum_{i=1}^n c_i \mathbf{V}_i$$

$$\text{LGC } \mathbf{f} \rightarrow \sum_{i=1}^n \frac{1}{(1 + \omega \lambda_i)} c_i \mathbf{V}_i$$

Low-pass on  $[0, \infty)$ ,  
continuous parameterization.

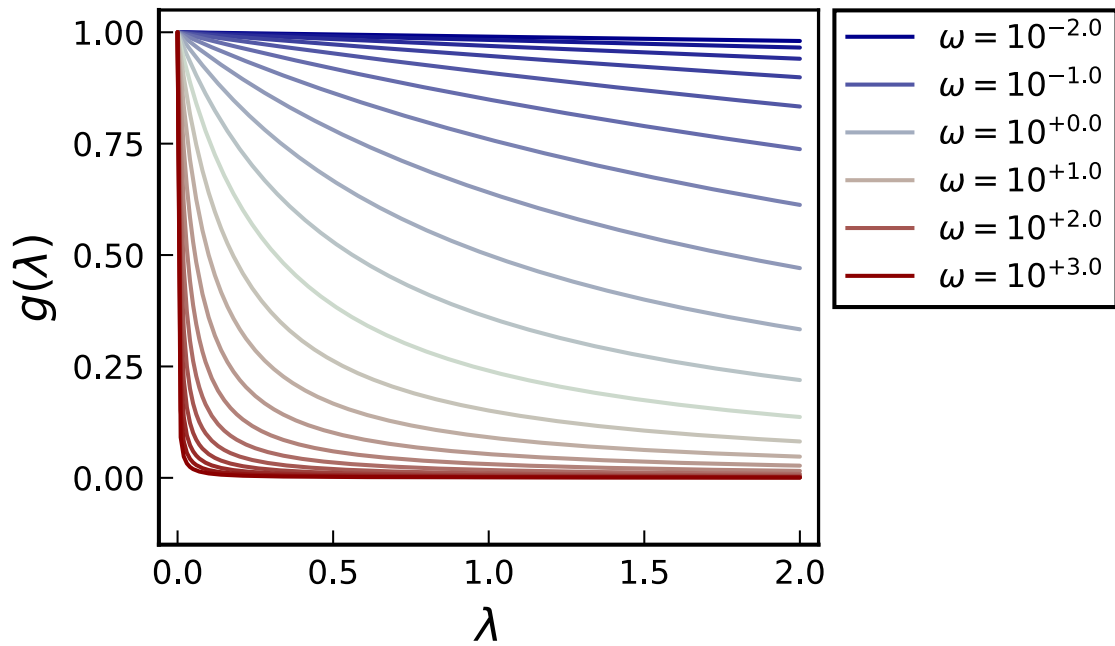
$$\text{SGC } \mathbf{f} \rightarrow \sum_{i=1}^n (1 - d/(d+1)\lambda_i)^K c_i \mathbf{V}_i$$

Low-pass on  $[0, (d + 1)/d]$ ,  
discrete parameterization.

} Encouraging  
smoothness.

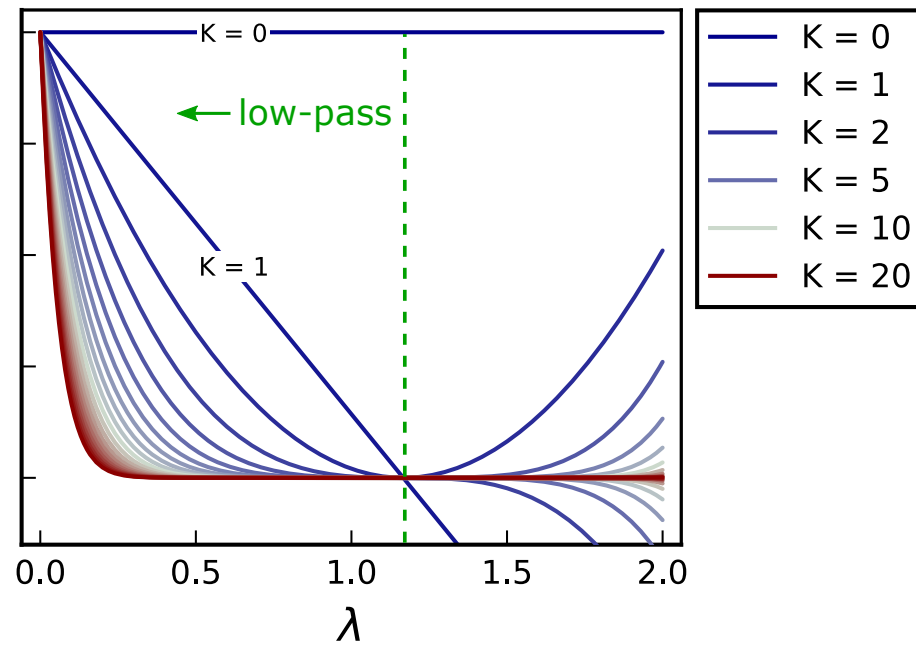
# Our model helps us understand smoothing.

LGC



$$\mathbf{f} = \sum_{i=1}^n c_i \mathbf{V}_i \rightarrow \sum_{i=1}^n \frac{1}{(1 + \omega \lambda_i)} c_i \mathbf{V}_i$$

SGC

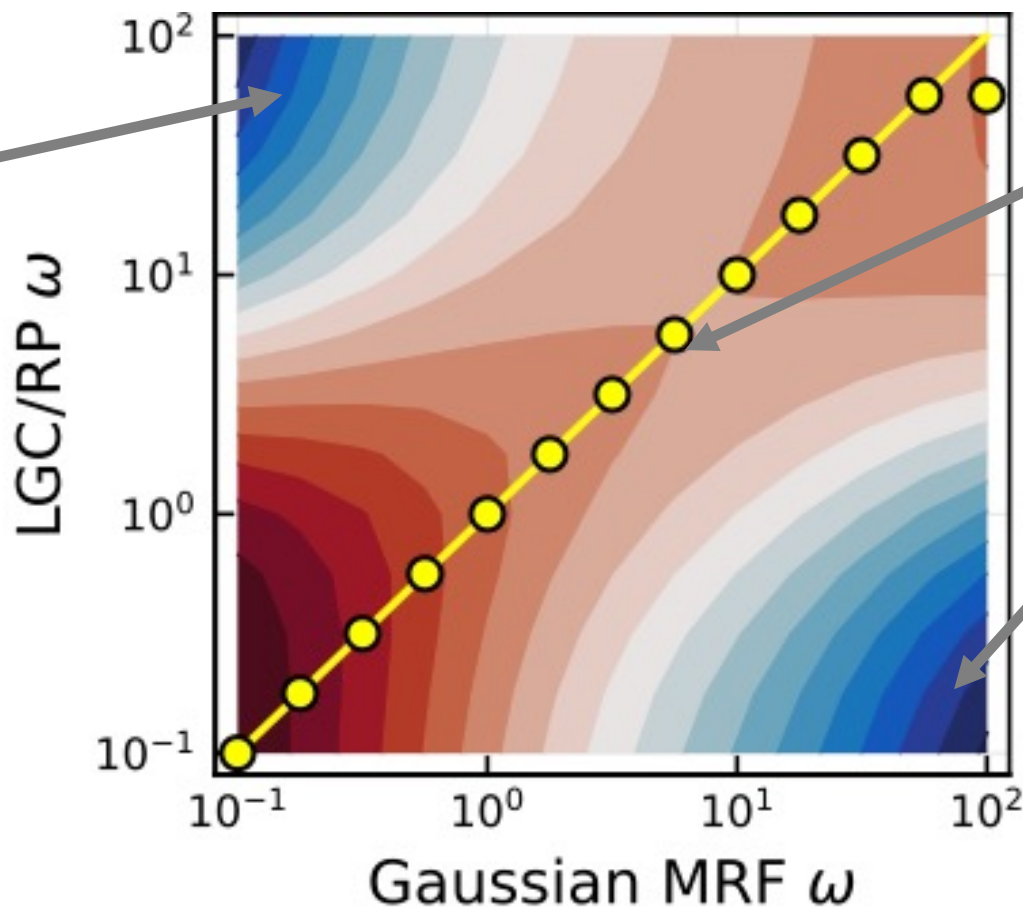


$$\mathbf{f} = \sum_{i=1}^n c_i \mathbf{V}_i \rightarrow \sum_{i=1}^n (1 - d/(d+1)\lambda_i)^K c_i \mathbf{V}_i$$

# Our model helps us understand smoothing.

“Oversmoothing.”

Empirically discussed  
problem with GNNs.  
[Li+ 18; Oono-Suzuki 20;  
Zhou-Akoglu 20]



cross validation can  
identify the model  
parameter.

Undersmoothing?

Possible but not  
discussed in the  
literature.

0.16  Test  $R^2$ .  
0.88



# We can also evaluate on our generative model.

	$h_0$	LP ( $\alpha$ )	LR	LGC ( $\alpha$ )	SGC ( $K$ )	GCN ( $K$ )	LGC/RP ( $\alpha$ )	SGC/RP ( $K, \alpha$ )	GCN/RP ( $K, \alpha$ )
Low homophily.	1	0.19 (0.79)	0.68	0.70 (0.28)	0.37 (1.8)	0.34 (1.7)	<b>0.73</b> (0.29)	0.40 (1.8, 0.21)	0.37 (1.7, 0.21)
	10	0.43 (0.95)	0.48	0.58 (0.57)	0.45 (2.1)	0.45 (2.0)	<b>0.68</b> (0.56)	0.56 (2.1, 0.46)	0.54 (2.0, 0.43)
High homophily.	100	0.59 (0.99)	0.24	0.42 (0.85)	0.38 (2.3)	0.45 (2.5)	<b>0.64</b> (0.85)	0.63 (2.3, 0.81)	0.62 (2.5, 0.79)

- GCN more expressive but prone to overfitting.
- More homophily  $\rightarrow$  larger  $K, \alpha$
- Adding residual prop never hurts!
- GCN better with more homophily?  
“memorizing” neighborhood features (zero training error)  
+ smoothness in data  $\rightarrow$  better out-of-sample prediction

# Our model provides a nice setup for inductive learning.

## Problem input.

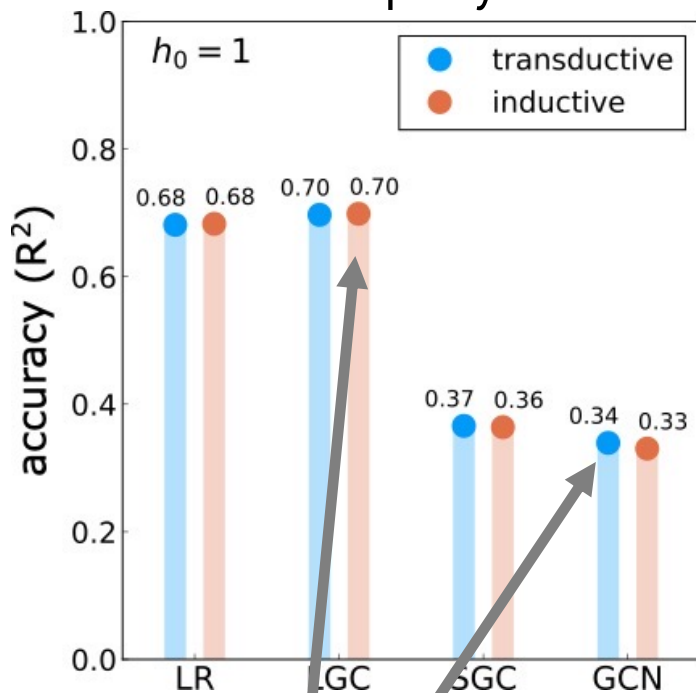
- Graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$ .
- $|V_1| \times p$  matrix  $\mathbf{X}_1$  and  $|V_2| \times p$  matrix  $\mathbf{X}_2$  of node features (same features)
- Subset  $L_1 \subset V$  of labeled nodes.
- Length- $|L_1|$  vector  $\mathbf{y}_{L_1}$  of outcomes on  $L_1$ .

## Problem output.

- Length- $|V_2|$  vector  $\mathbf{y}$  of outcomes on nodes  $V_2$ .

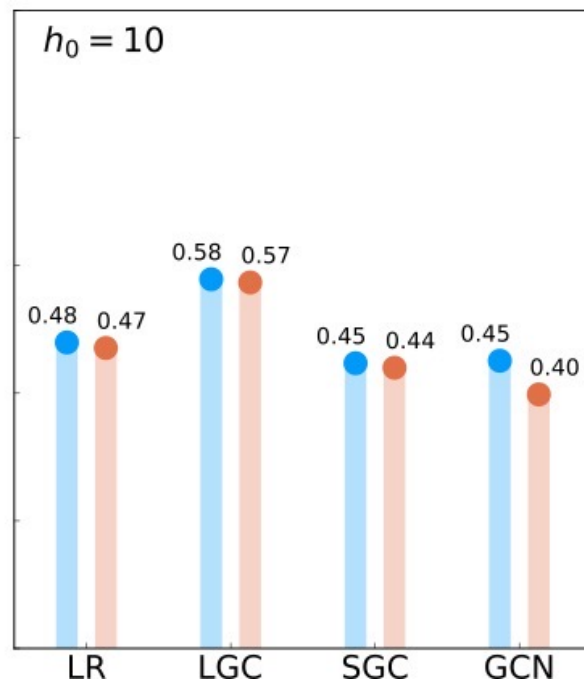
# Our model provides a nice setup for inductive learning.

Predictive features,  
low homophily.

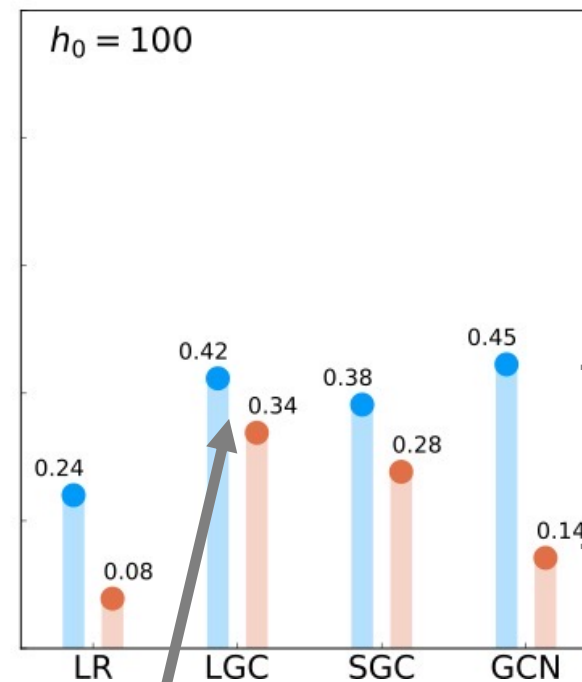


No performance  
degradation.

High homophily.

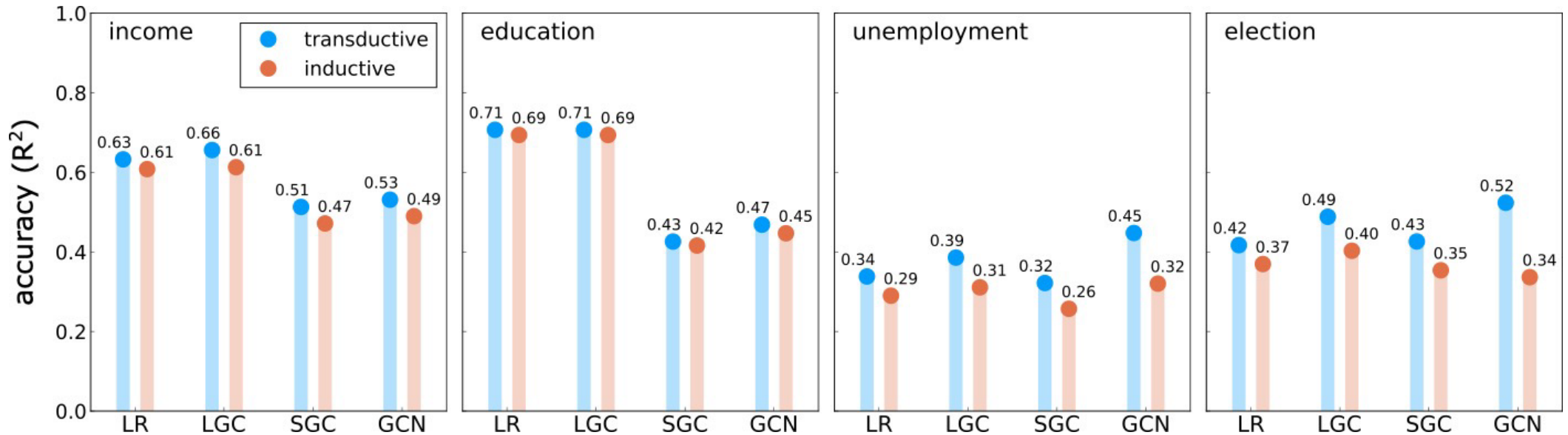


A bit of degradation.



Bad  
overfitting!

# Our model provides a nice setup for inductive learning.



- Graph  $G_1$  from 2012 election data.
- Graph  $G_2$  from 2016 election data.

# Major takeaway. Label propagation is a powerful tool.



1. LP can be applied to residuals (correlated errors).
2. LP can be applied to features (smoothing / de-noising).
3. While traditionally seen as separate ideas, LP and basic GNN ideas can be derived from a common model and combined effectively.
4. LP is scalable and easy to program. Just big SpMVs!
5. Linear models are often superior to nonlinear ones (GNNs) in practice... you just need to find the right one.

$$\mathbf{y}_U^{\text{LGC/RP}} = [(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \boldsymbol{\beta}]_U - (\mathbf{I} + \omega \mathbf{N})_{UU}^{-1} (\mathbf{I} + \omega \mathbf{N})_{UL} (\mathbf{y}_L - [(\mathbf{I}_n + \omega \mathbf{N})^{-1} \mathbf{X} \boldsymbol{\beta}]_L)$$


# There are lots of open research directions.

1. Theory or more principled approaches for classification?
2. More formal understanding of computational tradeoffs?
3. Similar ideas for other graph problems?  
link prediction, random walk prediction, graph classification, ...
4. Generative models to explain other GNN ideas?  
attention, GraphSAGE, skip connections, ...


# Label Propagation and Graph Neural Networks

**THANKS!** Austin R. Benson  
<http://cs.cornell.edu/~arb>  
 @austinbenson  
 arb@cs.cornell.edu

A Unifying Generative Model for Graph Learning Algorithms: Label Propagation, Graph Convolutions, and Combinations.  
Junteng Jia and Austin R. Benson. arXiv:2101.07730, 2021.

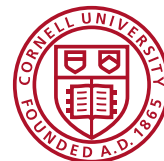
  <https://github.com/000Justin000/GaussianMRF>

Residual Correlation in Graph Neural Network Regression.  
Junteng Jia and Austin R. Benson. Proc. of KDD, 2020.

  <https://github.com/000Justin000/gnn-residual-correlation>

Combining Label Propagation and Simple Models Out-performs Graph Neural Networks.  
Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Proc. of ICLR, 2021.

  <https://github.com/CUAI/CorrectAndSmooth>



Cornell University