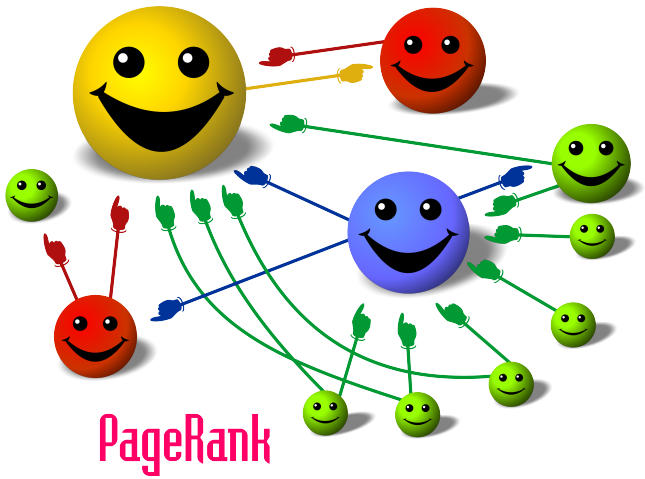
Estimate PageRank Scores - Using Graph Convolutional Neural Network

Yiming Gao, Yiqin Wu
Department of Statistics
Dec 11th, 2017

Agenda

Background



- Used to determine the importance of a webpage
- Considers links to be like votes
- Plays a central role in many of our web search tools

Problem Introduction

PageRank relies on the uniquely democratic nature of the Web by using its vast link structure as an indicator of an individual page's value.

Some factors:

- volumn of votes
- links a page receives
- the page that casts the vote

Problem Introduction

PageRank relies on the uniquely democratic nature of the Web by using its vast link structure as an indicator of an individual page's value.

Some factors:

- volumn of votes
- links a page receives
- the page that casts the vote

If the importance of everybody depends on everybody else, how can we calculate anything?

Google PageRank Algorithm

The answer is *iteration*.

Google provides its views on pages' relative importance. The importance of each page is the average of the importance of every page that points to it.

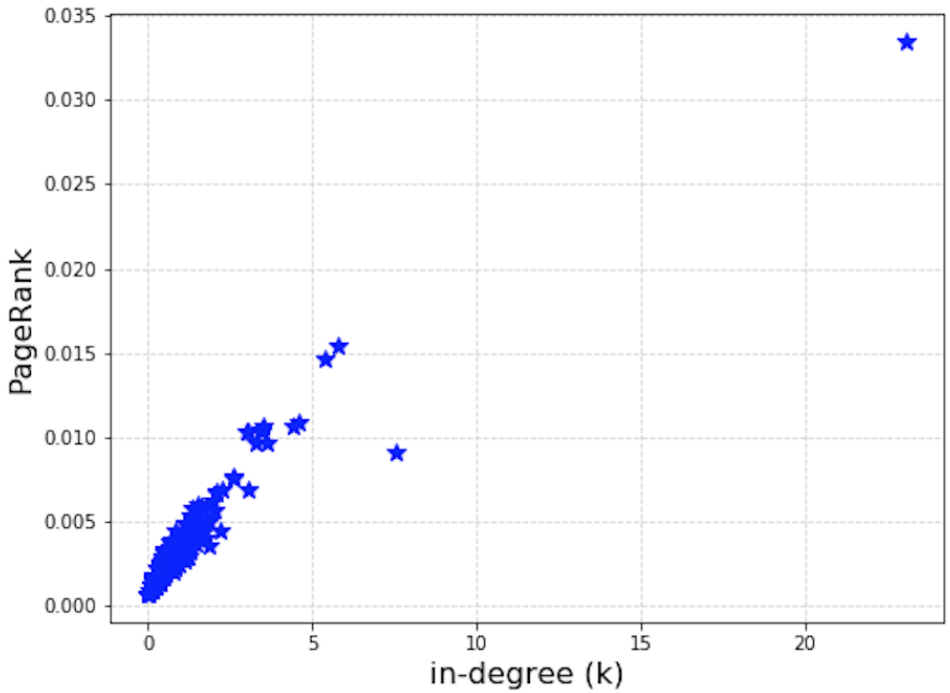
More mathematically, we recognize this as a Markov Chain:

$$\vec{P} = G\vec{P}$$

where P is the PageRank vector (the vector of "importances") and G is known as the "Google Matrix".

Google PageRank Algorithm

- generate the Google Matrix from the adjacency matrix
- PageRank vector vs. degree
- There is a strong, although not perfect, correlation

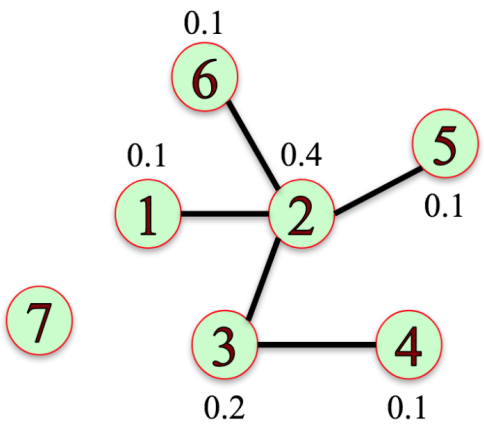


- Treat Google PageRank scores as ground truth
- **Damping factor**: the click-through probability, is included to prevent sinks (i.e. pages with no outgoing links) from "absorbing"the PageRanks of those pages connected to the sinks.
  - equals 1, end up in a sink
  - equals 0

Input Graphs

- Preferential Attachment Model
- Small World Model with Low Rewiring Probability
- Small World Model with High Rewiring Probability
- Stanford Web Data

Steps

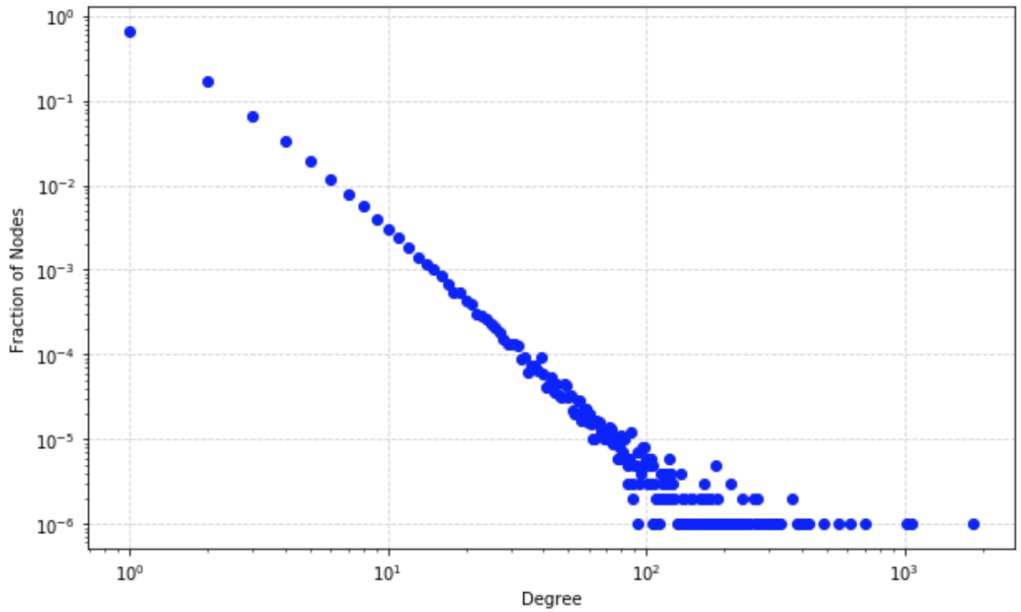


- Start with two nodes connected by an edge
- At each time step, add a new node with an edge connecting it to an existing node
- Choose the node to connect to at random with probability
- The probability of connecting to a node u of degree k_u is $\frac{k_u}{\sum_j k_j}$

Visualizations

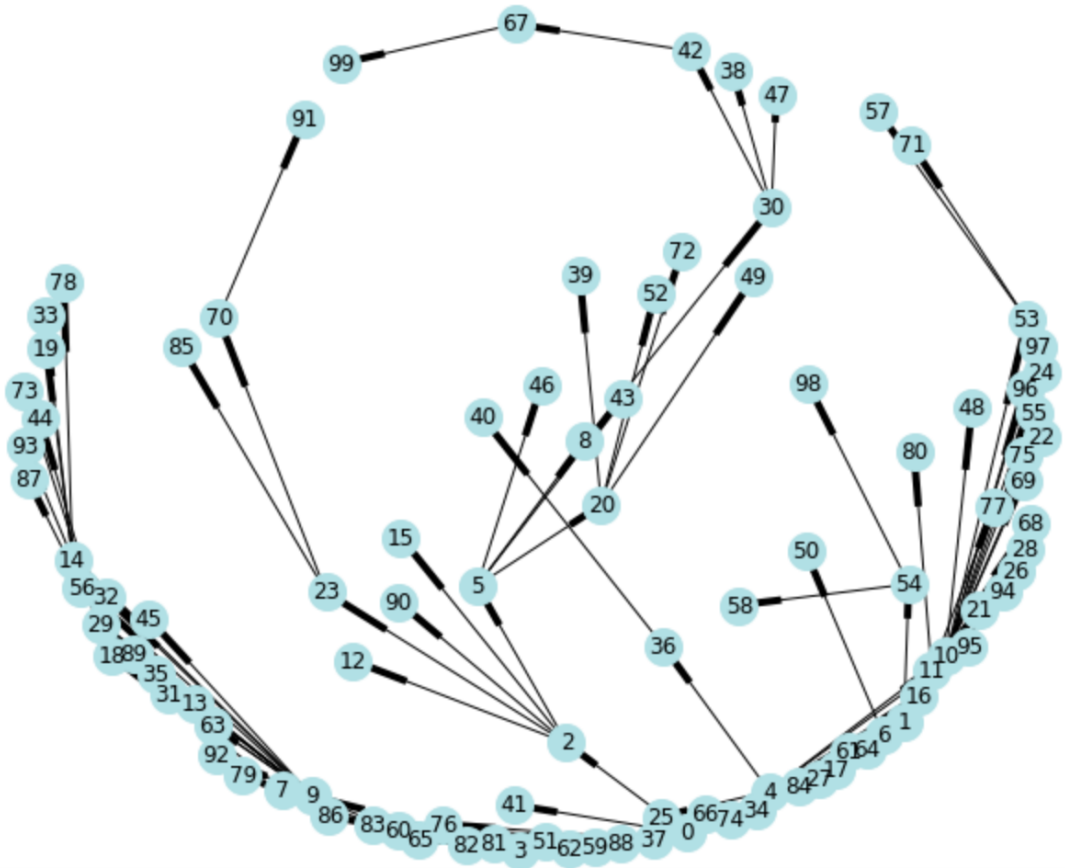
- Degree Distribution

The resulting degree distribution, the probability $P[k]$ of a node chosen uniformly at random having degree k is clearly broadtail with an exponent of approximately 3.



Visualizations

- Graph: Preferential Attachment Model with 100 Nodes



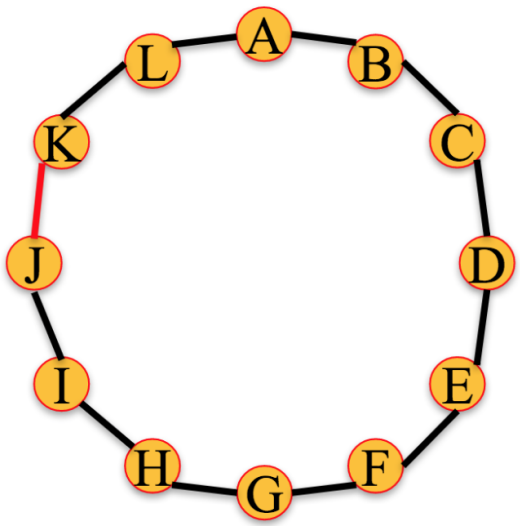
Motivation

- The world is small in the sense that "short" paths exists between almost any two people. Real networks exhibit high clustering coefficient and small average shortest paths.

Steps:

- Start with a ring of n nodes, where each node is connected to its k nearest neighbors
- Fix a parameter $p \in [0, 1]$
- Consider each edge (u, v). With probability p, select a node w at random and rewire the edge (u, v) so it becomes (u, w).

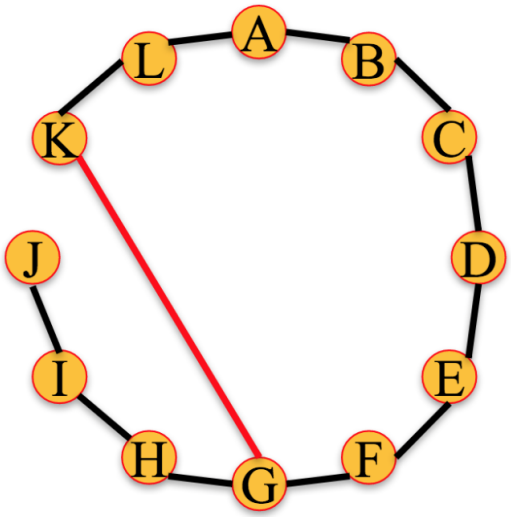
Steps



- Start with a ring of n nodes, where each node is connected to its k nearest neighbors
- Fix a parameter $p \in [0, 1]$
- Consider each edge (u, v). With probability p, select a node w at random and rewire the edge (u, v) so it becomes (u, w).

Example: $k = 2, p = 0.4$
Rewire!

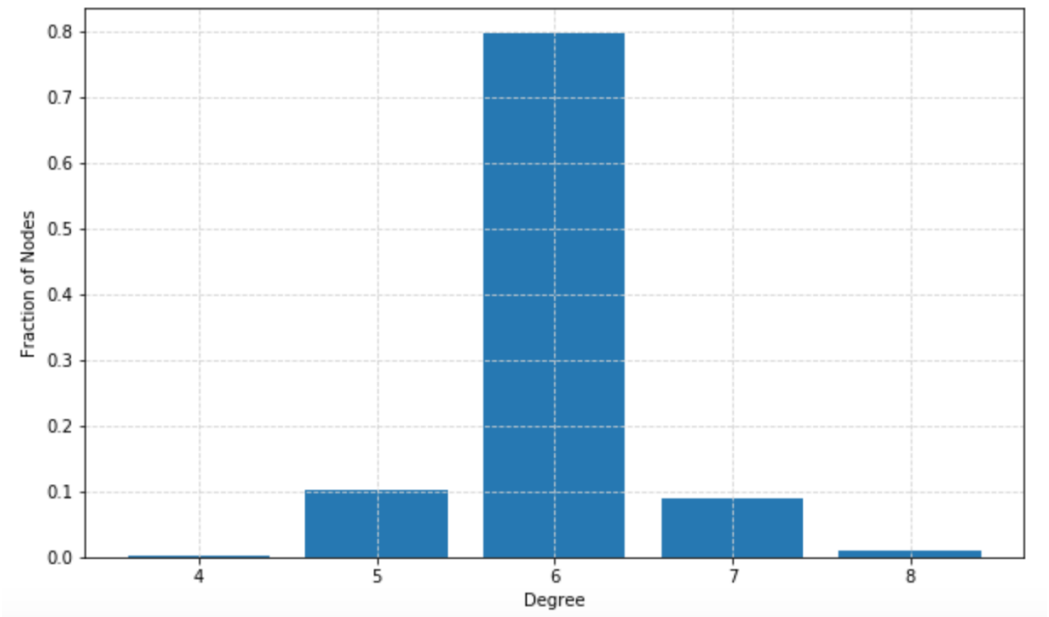
Steps



- Start with a ring of n nodes, where each node is connected to its k nearest neighbors
- Fix a parameter $p \in [0, 1]$
- Consider each edge (u, v). With probability p, select a node w at random and rewire the edge (u, v) so it becomes (u, w).

Example: $k = 2, p = 0.4$
Rewire!

Degree Distribution



This is a small world network with 1000 nodes, $k = 6$, and $p = 0.04$. There is no power law degree distribution.

Summary

- Real social networks appear to have small shortest paths between nodes and high clustering coefficient.
- The preferential attachment model produces networks with small shortest paths between nodes but **very small** clustering coefficient.
- It starts with a ring lattice with nodes connected to k nearest neighbors, and it rewires edges with probability p.
- Graph: Small World Model with $p = 0.05$

Dataset Information

Nodes represent pages from Stanford University (stanford.edu) and directed edges represent hyperlinks between them. The data was collected in 2002.

Dataset statistics	
Nodes	281903
Edges	2312497
Average Clustering Coefficient	0.5976
Diameter (longest shorest path)	674

Mean Square Error

$$Cost = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- known as L2 loss function that minimizes the squared differences between the estimated and existing targeting values
- labels: the ground truth output tensor
- predictions: the predicted outputs
- **not robust**
- most common for regression problems

Absolute Difference

$$Cost = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

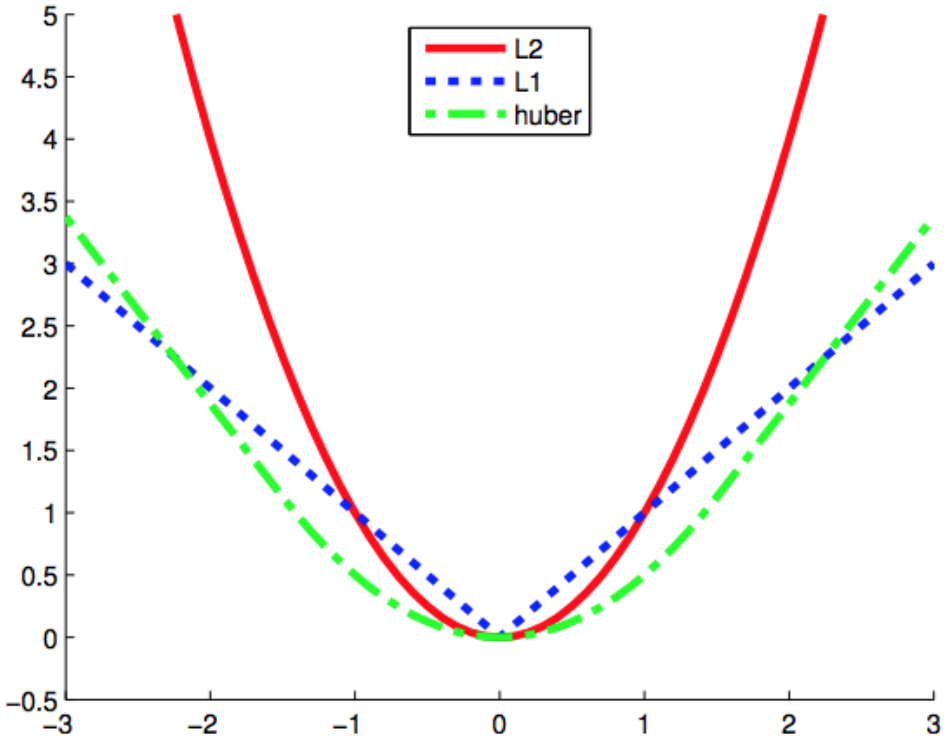
- known as L1 loss function that minimizes the absolute differences between the estimated values and the existing target values
- labels: the ground truth output tensor
- predictions: the predicted outputs
- **not differentiable**

Huber Loss

$$Cost = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- Quadratic for $|y - \hat{y}| \leq \delta$ and linear for $|y - \hat{y}| > \delta$
- labels: the ground truth output tensor
- predictions: the predicted outputs
- **robust and differentiable**

Comparison



Activation Function

The activation function is a rectified linear unit (ReLU), commonly used and highly useful.

Output Layer Loss Function

- In classification, we use cross-entropy if we have a sigmoid of softmax non-linearity in the output layer of our network (in homework)
- Our target is **continuous**, we use *Means Squared Error* (MSE) as the loss function.

Parameters

We train on a small world model with 1000 nodes and $p = 0.05$.

Dataset	Size
Training set	700
Validation set	100
Test set	200

- Epoch
- Learning rate
- Drop-out rate
- Loss (Cost) function

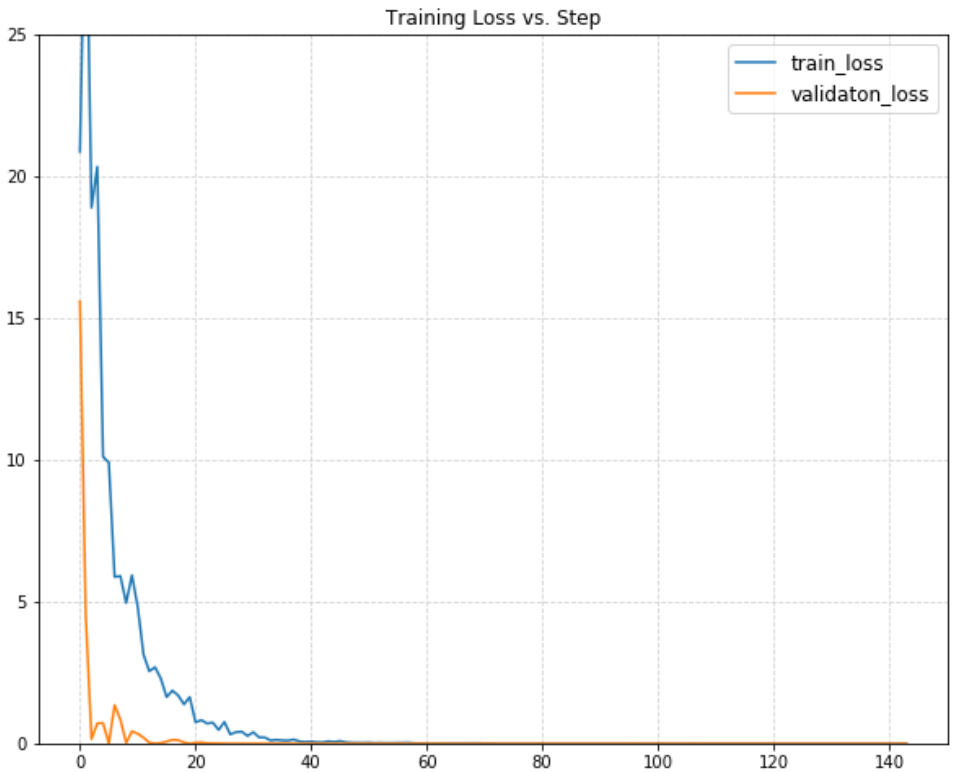
Learning Rate

Dataset	Small World Model ($p = 0.05$)
Learning Rate	0.01—0.05—0.1
Drop-out Rate	0.4
Cost Function	MSE
Adjacency Matrix	$\tilde{A}$
Weight Decay	Yes

- Learning rate = 0.01, stop at epoch 174, test MSE = 0.00001
- Learning rate = 0.05, stop at epoch 148, test MSE = 9.40e-07
- Learning rate = 0.1, stop at epoch 177, test MSE = 4.97-07

Learning Rate (Cont.)

Learning rate = 0.05



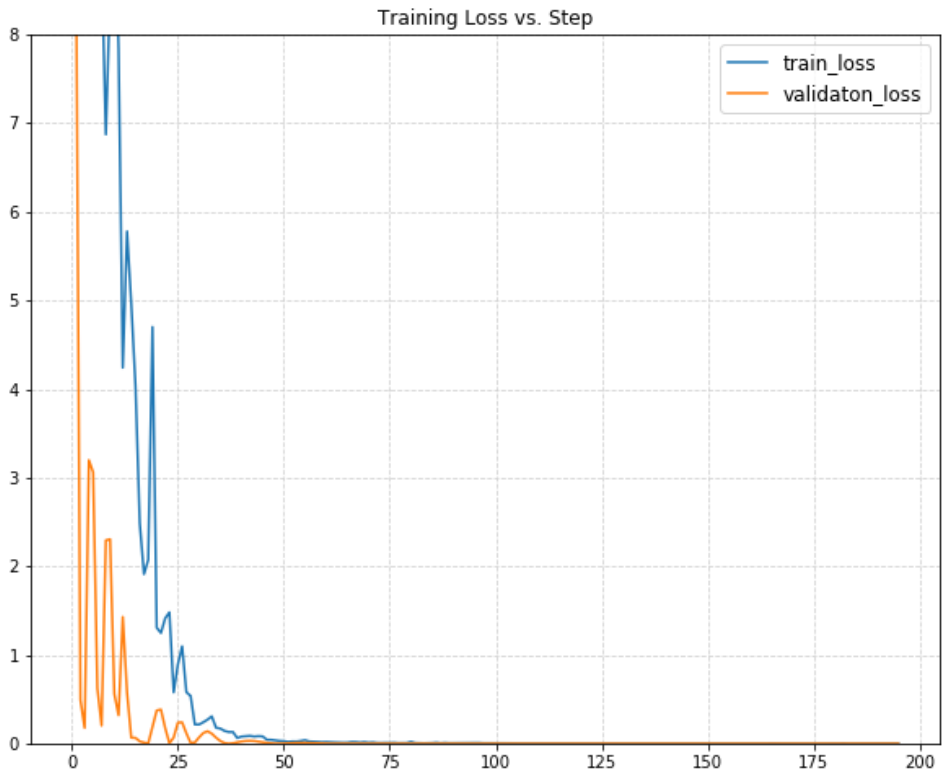
Drop-out Rate

Dataset	Small World Model ($p = 0.05$)
Learning Rate	0.05
Drop-out Rate	0.1 — 0.5
Cost Function	MSE
Adjacency Matrix	$\tilde{A}$
Weight Decay	Yes

- Drop-out Rate = 0.1, stop at epoch 221, test MSE = 4.41e-06
- Drop-out Rate = 0.5, stop at epoch 196, test MSE = 4.40e-07

Drop-out Rate (Cont.)

Drop-out rate = 0.5



Cost Function

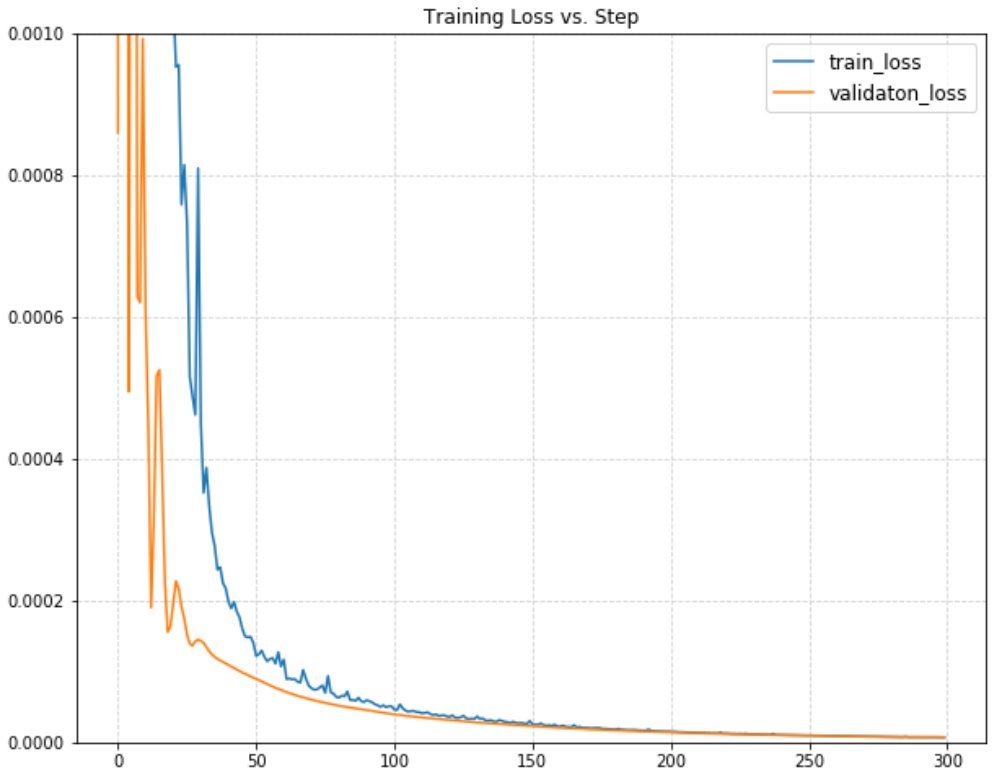
Dataset	Small World Model ($p = 0.05$)
Learning Rate	0.05
Drop-out Rate	0.5
Cost Function	MSE—Absolute Diff—Huber Loss
Adjacency Matrix	$\tilde{A}$
Weight Decay	Yes

- MSE (L2 Loss), stop at epoch 148, test MSE = 9.40e-07
- Absolute Difference (L1 Loss), stop at epoch 128, test MSE = 4.96e-07
- Huber Loss ($\delta = 5e-1$), stop at epoch 300, test MSE = 3.01e-07

For Huber loss, we try different δ values: 5e-3, 5e-2, 5e-1 and 1, and show the best result with $\delta = 5e-1$.

Cost Function (Cont.)

Huber Loss with $\delta = 5e - 1$.



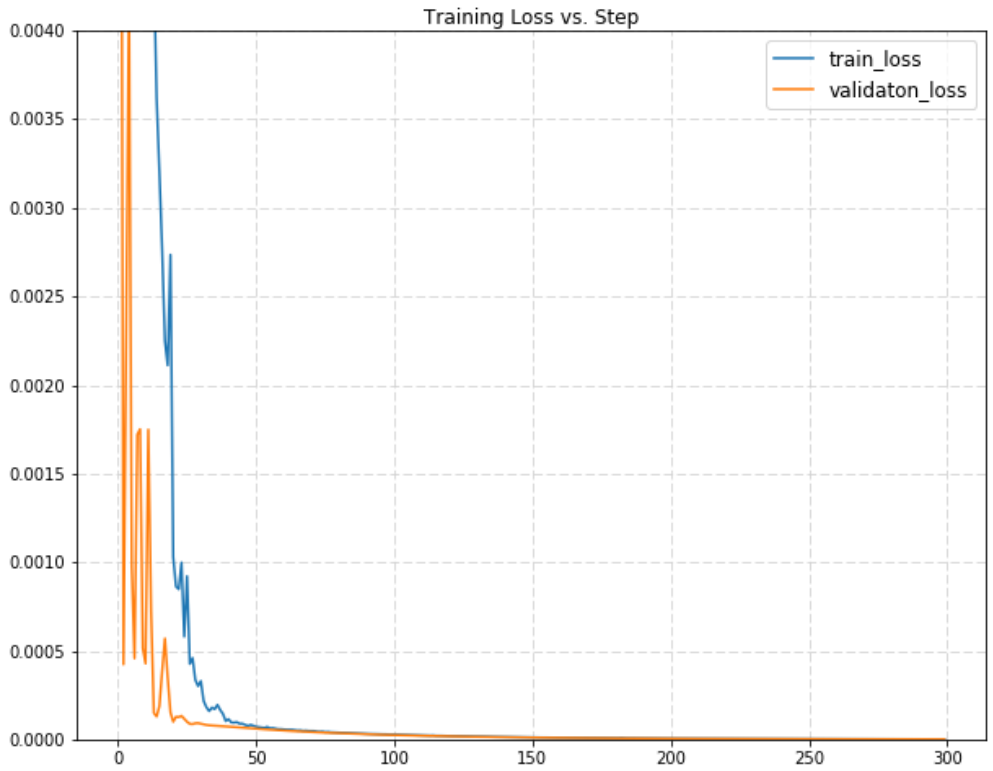
Weight Decay

Dataset	Small World Model ($p = 0.05$)
Learning Rate	0.05
Drop-out Rate	0.5
Cost Function	Huber Loss
Adjacency Matrix	$\tilde{A}$
Weight Decay	Yes

- With weight decay, stop at epoch 300, test MSE = 3.73e-07
- Without weight decay, stop at epoch 170, test MSE = 4.42e-07

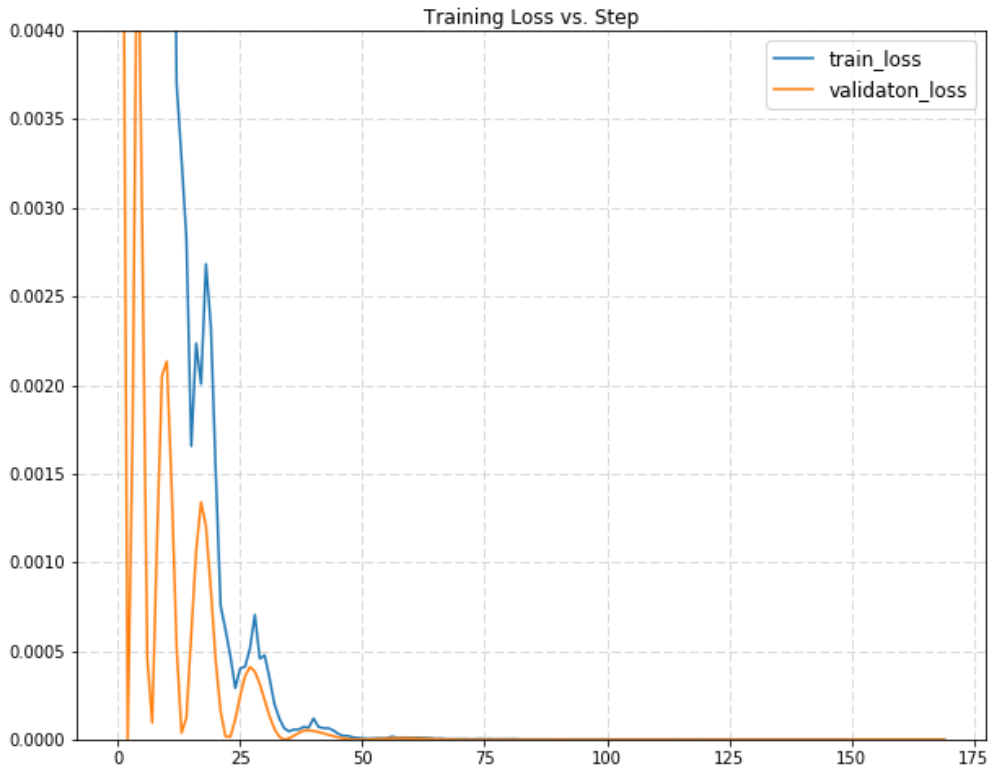
Weight Decay (Cont.)

With weight decay



Weight Decay (Cont.)

Without weight decay



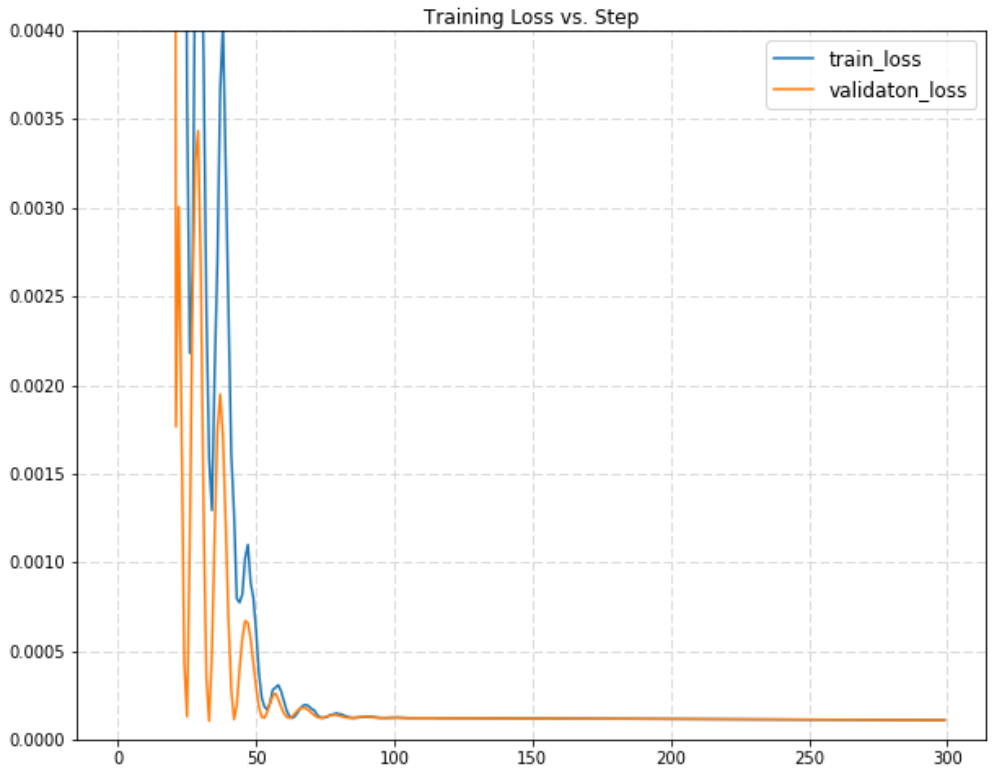
Adjacency Matrix

Dataset	Small World Model ($p = 0.05$)
Learning Rate	0.05
Drop-out Rate	0.5
Cost Function	Huber Loss
Adjacency Matrix	$\tilde{A}\|A^2\|\tilde{A}^2\|\tilde{A}^3$
Weight Decay	Yes

- Use $\tilde{A}$, stop at epoch 300, test MSE = 3.73e-07
- Use A^2, stop at epoch 300, test MSE = 3.76e-07
- Use $\tilde{A}^2$, stop at epoch 300, test MSE = 3.60e-07
- Use $\tilde{A}^3$, stop at epoch 300, test MSE = 3.89e-07

Adjacency Matrix (Cont.)

Adjacency matrix $= A^2$.



Does the Architecture Scale?

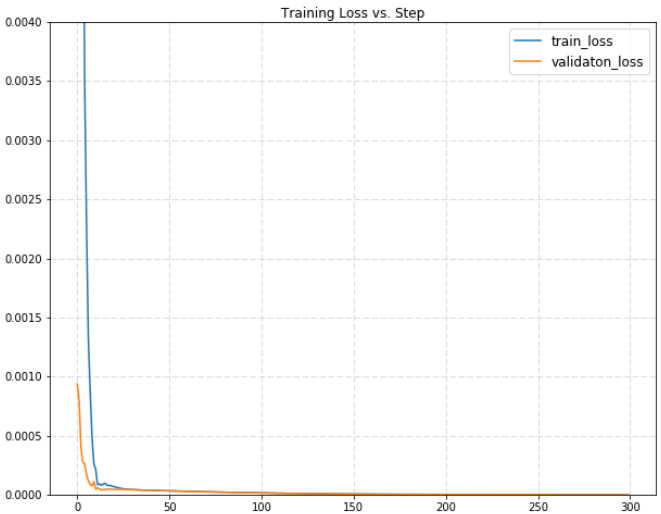
Train and validate on real dataset.

Nodes represent pages from Stanford University (stanford.edu) and directed edges represent hyperlinks between them. The data was collected in 2002.

Here we choose 11000 nodes randomly in this dataset to train and test the architecture. (Otherwise, OOM error)

Stanford Website

Train 10000, validate 1000.

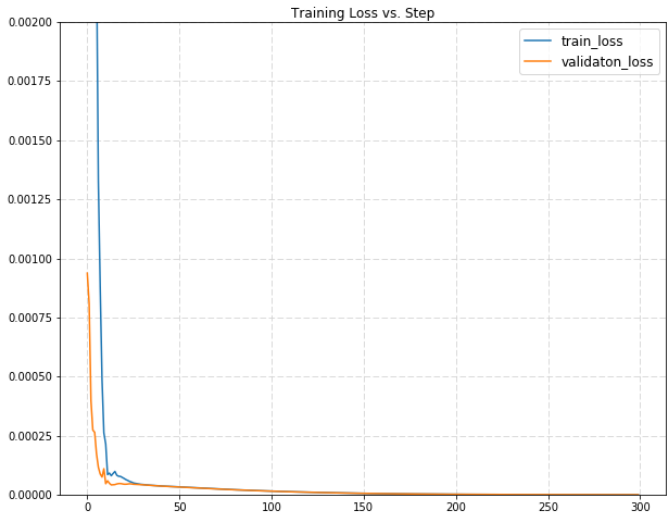


- Learning rate $= 0.05$
- Dropout rate $= 0.5$
- Cost function: Huber Loss
- Adjacency matrix: $\tilde{A}$

Train stop at epoch 300, MSE $=$ 3.03e-11.

Stanford Website

Train 2000, validate 8000.



- Learning rate $= 0.05$
- Dropout rate $= 0.5$
- Cost function: Huber Loss
- Adjacency matrix: $\tilde{A}$

Train stop at epoch 300, MSE $=$ 2.03e-11.

Future Work

- Train the network with more layers
- Compare with existing algorithms
- Sample size is relatively small compared to web data size in reality. If the network is faced with a wide scope attack and most of the nodes vanish, will the training of the architecture be successful?

Future Work

- Train the network with more layers
- Compare with existing algorithms
- Sample size is relatively small compared to web data size in reality. If the network is faced with a wide scope attack and most of the nodes vanish, will the training of the architecture be successful?

Questions?