GCN

Daniel Schwartz

June 2019

1 Graph Networks

1.1 Input

- Feature matrix NxF^0, X
- N is the number of nodes
- F^0 is the number of input features for each node
- NxN matrix representation of the graph structure (adjacency matrix)

1.2 Simple Propagation Rule

- $f(H^I, A) = \sigma(AH^IW^I)$
- W^I is the weight matrix for layer i (dimensions F^IxF^{i+1})
- σ is a non-linear activation function (ReLU function)
- Size of second dimension of weight matrix determines number of features at next layer

1.2.1 Issues with Propagation Rule

- Does not include its own features
- Representation only includes neighbor nodes unless has a self-loop
- Nodes with large degrees will have large values and small degrees will have smaller values in representation
- Causes vanishing or exploding gradients (problematic stochastic gradient descent)

1.2.2 Solving Propagation Rule Issues

- Add identity matrix to adjacency matrix before applying propagation rule to include its own features
- Normalize feature representations by multiplying adjacency matrix by its inverse degree matrix

1.3 Graph Neural Network Transition Functions

1.3.1 Branched Fixed Point Theorem and Jacobi Method

- There exists a unique solution to a system of equations and provides a method to compute those fixed points
- If assuming there is a metric space X and a mapping $T : X \to X$
- Mapping is a contraction mapping on X in which T admits a unique fixed point x^* in X (e.g. $T(x^*) = x^*$)
- GNN transition function is assumed to be a contractive mapping with respect to the nodes' state
- Since BFP guarantees a unique solution, Jacobi iterative method is used to solve the fixed point solution, the nodes' state
- Solved by approximating values to plug in and then iterating until convergence

1.3.2 Message Passing Neural Network

- Does not assume edge features are discrete
- Message Phase
  - Transition function and output function combined where M is the transition function and U is the output function
  - $m_n^{t+1} = \sum_{u \in ne[n]} M_t(h_n^t, h_u^t, e_{(n,u)})$
  - $h_n^{t+1} = U_t(h_n^t, m_n^{t+1})$
- Readout Phase
  - Function of all the nodes' states and outputs a label for the entire graph
  - $\hat{y} = R(\{h_n^T | n \in G\})$

1.4 Spectral Graph Convolutions

- $f(X, A) = \sigma(D^{-0.5}\hat{A}D^{-0.5}XW)$
- Normalizes aggregate using the degree matrix D raised to a negative power (asymmetric)
- Considers degree of ith node but also degree of jth node
- Weighs neighbor in the weighted sum higher if they have a lower degree and lower if they have a high-degree

1.4.1 Semi-Supervised Classification with GCNs

- Make use of both labeled and unlabeled examples
- Nodes that share neighbors tend to have similar feature representations
- Train GCN on labeled nodes and effectively propagate node label info to unlabeled nodes b updating weight matrices that are shared across all nodes:
  - Perform forward propagation through GCN
  - Apply sigmoid function row-wise on the last layer in the GCN
  - Compute cross entropy loss on known node labels
  - Backpropogate loss and update weight matrices W in each layer

2 Capsule Network Implementation Notes

- Gets batch data
- Creates one hot of labels
- Builds Architecture
  - Convolution Layer 1
  - Primary Capsule Layer
  - Digit Caps Layer
    - Routing
  - Masking
    - Calculates magnitude of v_c and then applies softmax
    - Calculates index of max softmax of 10 caps
  - Reconstruct MNIST images with 3 Fully Convolutional Layers
- Loss function
  - Margin loss

- Reconstruction loss
- Total loss (= Margin loss + Regularization scale $*$ Reconstruction loss)