Deep Learning on Graphs with Graph Convolutional Networks

Thomas Kipf, 6 April 2017
joint work with Max Welling (University of Amsterdam)
The success story of deep learning

Speech data

Natural language processing (NLP)

...
The success story of deep learning

Deep neural nets that exploit:
- translation invariance (weight sharing)
- hierarchical compositionality
Recap: Deep learning on Euclidean data

Euclidean data: grids, sequences...

2D grid
Recap: Deep learning on Euclidean data

Euclidean data: grids, sequences...

- 2D grid
- 1D grid
Recap: Deep learning on Euclidean data

Convolutional neural networks (CNNs)

(Animation by Vincent Dumoulin)
Recap: Deep learning on Euclidean data

Convolutional neural networks (CNNs)

(Animation by Vincent Dumoulin)

(Source: Wikipedia)
Recap: Deep learning on Euclidean data

Convolutional neural networks (CNNs)

Recurrent neural networks (RNNs)

(Source: Christopher Olah’s blog)
Traditional vs. “deep” learning

Traditional approach

Hand-designed feature extractor → Classifier “on top” → Output
Traditional vs. “deep” learning

Traditional approach

End-to-end learning
CNNs: Message passing on a grid-graph

Single CNN layer with 3x3 filter:
CNNs: Message passing on a grid-graph

Single CNN layer with 3x3 filter:

(Animation by Vincent Dumoulin)
CNNs: Message passing on a grid-graph

Single CNN layer with 3x3 filter:

(Animation by Vincent Dumoulin)
CNNs: Message passing on a grid-graph

Single CNN layer with 3x3 filter:

Update for a single pixel:
- Transform messages individually $W_i h_i$
- Add everything up $\sum_i W_i h_i$
CNNs: Message passing on a grid-graph

Single CNN layer with 3x3 filter:

Update for a single pixel:
- Transform messages individually $W_i h_i$
- Add everything up $\sum_i W_i h_i$

Full update:
$$h_4^{(l+1)} = \sigma \left( W_0^{(l)} h_0^{(l)} + W_1^{(l)} h_1^{(l)} + \cdots + W_8^{(l)} h_8^{(l)} \right)$$
Graph-structured data

What if our data looks like this?
Graph-structured data

What if our data looks like this?

or this:
Graph-structured data

What if our data looks like this?

Real-world examples:
- Social networks
- World-wide-web
- Protein-interaction networks

or this:

- Telecommunication networks
- Knowledge graphs
- …
Graph-structured data

Graph: \( G = (\mathcal{V}, \mathcal{E}) \)

Adjacency matrix: \( A \)

\[
A = \begin{pmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0
\end{pmatrix}
\]
Graph-structured data

Graph: \( G = (\mathcal{V}, \mathcal{E}) \)

Adjacency matrix: \( A \)

Model wish list:

- Trainable in \( O(|\mathcal{E}|) \) time
- Applicable even if the input graph changes
A naïve approach

- Take adjacency matrix $A$ and feature matrix $X$
- Concatenate them $X_{in} = [A, X]$
- Feed them into deep (fully connected) neural net
- Done?

![Graph representation and neural network diagram]
A naïve approach

• Take adjacency matrix $A$ and feature matrix $X$
• Concatenate them $X_{in} = [A, X]$
• Feed them into deep (fully connected) neural net
• Done?

Problems:

• Huge number of parameters $O(N)$
• Re-train if graph changes
A naïve approach

- Take adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$
- Concatenate them $\mathbf{X}_{\text{in}} = [\mathbf{A}, \mathbf{X}]$
- Feed them into deep (fully connected) neural net
- Done?

Problems:
- Huge number of parameters $\mathcal{O}(N)$
- Re-train if graph changes
A naïve approach

• Take adjacency matrix \( A \) and feature matrix \( X \)
• Concatenate them \( X_{\text{in}} = [A, X] \)
• Feed them into deep (fully connected) neural net
• Done?

We need weight sharing!

\[ \rightarrow \text{CNNs on graphs or} \]

“Graph Convolutional Networks” (GCNs)

Problems:

• Huge number of parameters \( O(N) \)
• Re-train if graph changes
GCNs with 1st-order message passing

(related idea was first proposed in Scarselli et al. 2009)

Consider this undirected graph:
GCNs with 1st-order message passing

(related idea was first proposed in Scarselli et al. 2009)

Consider this undirected graph:

Calculate update for node in red:
Consider this undirected graph:

Calculate update for node in red:
GCNs with 1st-order message passing
(related idea was first proposed in Scarselli et al. 2009)

Consider this undirected graph:

Calculate update for node in red:

Update rule:

\[ h_i^{(l+1)} = \sigma \left( h_i^{(l)} W_0^{(l)} + \sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} h_j^{(l)} W_1^{(l)} \right) \]

\( \mathcal{N}_i \): neighbor indices
\( c_{ij} \): norm. constant (per edge)

Note: We could also choose simpler or more general functions over the neighborhood
GCN model architecture

Input: Feature matrix $X \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{A}$

$$X = H^{(0)}$$

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

[Kipf & Welling, ICLR 2017]
What does it do? An example.

Forward pass through **untrained** 3-layer GCN model

Parameters initialized randomly

$$f(\quad)$$

[Karate Club Network]

$$= \quad$$

2-dim output per node
Relation to Weisfeiler-Lehman algorithm

A “classical” approach for node feature assignment

Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

**Input:** Initial node coloring \((h_1^{(0)}, h_2^{(0)}, \ldots, h_N^{(0)})\)

**Output:** Final node coloring \((h_1^{(T)}, h_2^{(T)}, \ldots, h_N^{(T)})\)

\[ t \leftarrow 0; \]

repeat

\begin{align*}
\text{for } v_i \in V & \text{ do} \\
& h_i^{(t+1)} \leftarrow \text{hash} \left( \sum_{j \in N_i} h_j^{(t)} \right); \\
& t \leftarrow t + 1; \\
\text{until stable node coloring is reached;}
\end{align*}
Relation to Weisfeiler-Lehman algorithm

A “classical” approach for node feature assignment

Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

**Input**: Initial node coloring \((h_1^{(0)}, h_2^{(0)}, \ldots, h_N^{(0)})\)

**Output**: Final node coloring \((h_1^{(T)}, h_2^{(T)}, \ldots, h_N^{(T)})\)

\(t \leftarrow 0;\)

**repeat**

\[ \text{for } v_i \in V \text{ do} \]

\[ h_i^{(t+1)} \leftarrow \text{hash} \left( \sum_{j \in N_i} h_j^{(t)} \right); \]

\(t \leftarrow t + 1;\)

**until** stable node coloring is reached;
Relation to Weisfeiler-Lehman algorithm

A “classical” approach for node feature assignment

Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

Input: Initial node coloring \((h_1^{(0)}, h_2^{(0)}, \ldots, h_N^{(0)})\)
Output: Final node coloring \((h_1^{(T)}, h_2^{(T)}, \ldots, h_N^{(T)})\)

\[
t \leftarrow 0;
\]
repeat
\[
\text{for } v_i \in V \text{ do}
\]
\[
h_i^{(t+1)} \leftarrow \text{hash} \left( \sum_{j \in N_i} h_j^{(t)} \right);
\]
\[
t \leftarrow t + 1;
\]
until stable node coloring is reached;

Useful as graph isomorphism check for \textit{most} graphs

(exception: highly regular graphs)
Relation to Weisfeiler-Lehman algorithm

A “classical” approach for node feature assignment

Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

**Input:** Initial node coloring \((h_1^{(0)}, h_2^{(0)}, \ldots, h_N^{(0)})\)

**Output:** Final node coloring \((h_1^{(T)}, h_2^{(T)}, \ldots, h_N^{(T)})\)

\[
t \leftarrow 0;
\]

repeat

\[
\text{for } v_i \in V \text{ do}
\]

\[
h_i^{(t+1)} \leftarrow \text{hash} \left( \sum_{j \in N_i} h_j^{(t)} \right);
\]

\[
t \leftarrow t + 1;
\]

until stable node coloring is reached;

Useful as graph isomorphism check for *most* graphs

(exception: highly regular graphs)
Relation to Weisfeiler-Lehman algorithm

A “classical” approach for node feature assignment

Algorithm 1: WL-1 algorithm (Weisfeiler & Lehmann, 1968)

**Input:** Initial node coloring \((h_1^{(0)}, h_2^{(0)}, \ldots, h_N^{(0)})\)

**Output:** Final node coloring \((h_1^{(T)}, h_2^{(T)}, \ldots, h_N^{(T)})\)

1. \(t \leftarrow 0;\)
2. repeat
   1. for \(v_i \in \mathcal{V}\) do
      1. \(h_i^{(t+1)} \leftarrow \text{hash}(\sum_{j \in \mathcal{N}_i} h_j^{(t)});\)
3. until

**GCN:**

\[
h_i^{(l+1)} = \sigma \left( \sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} h_j^{(l)} W_1^{(l)} \right)
\]

Useful as graph isomorphism check for *most* graphs

(exception: highly regular graphs)
Semi-supervised classification on graphs

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

**Task:**
Predict node label of unlabeled nodes
Semi-supervised classification on graphs

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

**Task:**
Predict node label of unlabeled nodes

**Standard approach:**
graph-based regularization \[ [Zhu et al., 2003] \]

\[
\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{\text{reg}} \quad \text{with} \quad \mathcal{L}_{\text{reg}} = \sum_{i,j} A_{ij} \| f(X_i) - f(X_j) \|^2
\]

assumes: connected nodes likely to share same label
Semi-supervised classification on graphs

Embedding-based approaches

Two-step pipeline:

1) Get embedding for every node
2) Train classifier on node embedding

Examples: DeepWalk [Perozzi et al., 2014], node2vec [Grover & Leskovec, 2016]
Semi-supervised classification on graphs

**Embedding-based approaches**

Two-step pipeline:

1) Get embedding for every node
2) Train classifier on node embedding

**Examples**: DeepWalk [Perozzi et al., 2014], node2vec [Grover & Leskovec, 2016]

**Problem**: Embeddings are not optimized for classification!
Semi-supervised classification on graphs

Embedding-based approaches

Two-step pipeline:
1) Get embedding for every node
2) Train classifier on node embedding

Examples: DeepWalk [Perozzi et al., 2014], node2vec [Grover & Leskovec, 2016]

Problem: Embeddings are not optimized for classification!

Idea: Train graph-based classifier end-to-end using GCN

Evaluate loss on labeled nodes only:

\[
\mathcal{L} = - \sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}
\]

\(\mathcal{Y}_L\) set of labeled node indices
\(Y\) label matrix
\(Z\) GCN output (after softmax)
Toy example (semi-supervised learning)

Toy example (semi-supervised learning)

Video also available here: http://tkipf.github.io/graph-convolutional-networks
Application: Classification on citation networks

**Input**: Citation networks (nodes are papers, edges are citation links, optionally bag-of-words features on nodes)

**Target**: Paper category (e.g. stat.ML, cs.LG, …)

(Figure from: Bronstein, Bruna, LeCun, Szlam, Vandergheynst, 2016)
Experiments and results

**Model:** 2-layer GCN \[ Z = f(X, A) = \text{softmax} \left( \hat{A} \text{ReLU} \left( \hat{A}XW^{(0)} \right) W^{(1)} \right) \]

**Dataset statistics**

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

(Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017)
Experiments and results

Model: 2-layer GCN \[ Z = f(X, A) = \text{softmax}\left( \tilde{A} \text{ReLU}\left( \tilde{A}XW^{(0)} \right) W^{(1)} \right) \]

Dataset statistics

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

Classification results (accuracy)

<table>
<thead>
<tr>
<th>Method</th>
<th>Citeseer</th>
<th>Cora</th>
<th>Pubmed</th>
<th>NELL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ManiReg [3]</td>
<td>60.1</td>
<td>59.5</td>
<td>70.7</td>
<td>21.8</td>
</tr>
<tr>
<td>SemiEmb [24]</td>
<td>59.6</td>
<td>59.0</td>
<td>71.1</td>
<td>26.7</td>
</tr>
<tr>
<td>LP [27]</td>
<td>45.3</td>
<td>68.0</td>
<td>63.0</td>
<td>26.5</td>
</tr>
<tr>
<td>DeepWalk [18]</td>
<td>43.2</td>
<td>67.2</td>
<td>65.3</td>
<td>58.1</td>
</tr>
<tr>
<td>Planetoid* [25]</td>
<td>64.7 (26s)</td>
<td>75.7 (13s)</td>
<td>77.2 (25s)</td>
<td>61.9 (185s)</td>
</tr>
<tr>
<td>GCN (this paper)</td>
<td>70.3 (7s)</td>
<td>81.5 (4s)</td>
<td>79.0 (38s)</td>
<td>66.0 (48s)</td>
</tr>
<tr>
<td>GCN (rand. splits)</td>
<td>67.9 ± 0.5</td>
<td>80.1 ± 0.5</td>
<td>78.9 ± 0.7</td>
<td>58.4 ± 1.7</td>
</tr>
</tbody>
</table>

(Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017)
Experiments and results

**Model:** 2-layer GCN \[ Z = f(X, A) = \text{softmax} \left( \hat{A} \ \text{ReLU} \left( \hat{A} X W^{(0)} \right) W^{(1)} \right) \]

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No input features</th>
<th>DeepWalk [18]</th>
<th>Planetoid* [25]</th>
<th>GCN (this paper)</th>
<th>GCN (rand. splits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citeseer</td>
<td>43.2 (67.2)</td>
<td>65.3</td>
<td>77.2</td>
<td>79.0</td>
<td>78.9 ± 0.7</td>
</tr>
<tr>
<td>Cora</td>
<td>67.2</td>
<td>65.3</td>
<td>77.2</td>
<td>79.0</td>
<td>58.4 ± 1.7</td>
</tr>
<tr>
<td>Pubmed</td>
<td>64.7 (75.7)</td>
<td>77.2 (25s)</td>
<td>79.0 (38s)</td>
<td>66.0 (48s)</td>
<td></td>
</tr>
<tr>
<td>NELL</td>
<td></td>
<td>61.9 (185s)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017)
Other recent applications

Molecules

[Duvenaud et al., NIPS 2015]

Shapes

[Polar coordinates $\rho, \theta$]

[MoNet]

[Monti et al., 2016]

Knowledge Graphs

[Mikhail Baryshnikov]

U.S.A.

Vaganova Academy

:Ballet_dancer

:University

:Country

:Award

:Vilcek Prize

[Schlichtkrull et al., 2017]
Link prediction with Graph Auto-Encoders
Kipf & Welling, NIPS Bayesian Deep Learning Workshop, 2016

Encoder \[ Z = \text{GCN}(X, A) \]

Decoder \[ \hat{A} = \sigma(ZZ^T) \]
Further reading

Blog post Graph Convolutional Networks:  
http://tkipf.github.io/graph-convolutional-networks

Code on Github:  
http://github.com/tkipf/gcn

Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017:  
https://arxiv.org/abs/1609.02907

Kipf & Welling, Variational Graph Auto-Encoders, NIPS BDL Workshop, 2016:  
https://arxiv.org/abs/1611.07308

You can get in touch with me via:

- E-Mail: T.N.Kipf@uva.nl
- Twitter: @thomaskipf
- Web: http://tkipf.github.io