Power and Limitations of Graph Neural Networks

Seminar for Deep Neural Networks

Johannes Weidenfeller

Supervisor
Karolis Martinkus

8 March 2022
Content

1. Graph Neural Networks
2. Weisfeiler Lehman Isomorphism Test
3. GNNs with port numbering
4. Distributed computing
5. Communication capacity
6. Oversquashing
7. Conclusion
Graph Neural Networks
Graph Neural Networks

Basic definition

- Input: Graph $G = (V, E)$ with graph labels $x_v$ for each vertex $v \in V$

- Each layer consists of two steps\(^1\)

  1) Aggregate features of neighboring vertices

  \[ a_v^t = AGGREGATE^t \left( \{ x_u^t | u \in \mathcal{N}(v) \} \right) \]

  Examples: Sum, Mean, Max, MLPs

  \[
  \begin{align*}
  x_1^t & \rightarrow a_v^t \\
  x_2^t & \rightarrow a_v^t \\
  x_3^t & \rightarrow a_v^t
  \end{align*}
  \]

  2) Combine aggregate with current vertex label

  \[ x_v^{t+1} = COMBINE^t \left( x_v^t, a_v^t \right) \]

  Examples: Concatenation + Linear Mapping
Graph Neural Networks

Layer function

- We can combine the aggregate and combine functions to a single layer function $f_\theta$

Figure 1: Propagation of information in a graph neural network
Graph Neural Networks

Classification

- Depending on layer function we can distinguish between different GNN classes with different computational complexity\(^2\)

<table>
<thead>
<tr>
<th>Model</th>
<th>Set broadcasting GNNs</th>
<th>Multiset broadcasting GNNs</th>
<th>Vector-vector consistent GNNs</th>
<th>GNNs with access to unique vertex IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Input" /></td>
<td><img src="image" alt="Set" /></td>
<td><img src="image" alt="Multiset" /></td>
<td><img src="image" alt="Multiset and port numbering" /></td>
<td><img src="image" alt="Unique vertex IDs" /></td>
</tr>
</tbody>
</table>

\(v\), \(1\), \(2\), \(3\)
Graph Neural Networks

Readout function

- Often, we are interested in graph level classification/regression tasks

- GNNs can be extended through a **READOUT** function that combines features from all nodes

  \[ x_G = \text{READOUT} \left( \{ x_v^T \mid v \in V \} \right) \]

  (where \( T \) denotes the index of the last layer)

- Should be permutation invariant

- Examples: Summation, Mean/Max-Pooling

Figure 2: **READOUT** function in a graph neural network
Graph Neural Networks
Depth and Width

Definition 1

The depth $d$ of a Graph Neural Networks is equal to its number of layers.

Definition 2

The width $w$ of a Graph Neural Network is equal to the largest dimension of $x_v^t$ for any vertex $v$ and layer $t$

$$w = \max_{v \in V} \max_{t \in \{0, \ldots, d\}} \dim(x_v^t).$$

• The depth and width of a GNN play a crucial role in its computational power
The Weisfeiler Leman Isomorphism Test
Graph isomorphism

**Definition**

Two labeled graphs \( G = (V, E, X) \) and \( G' = (V', E', X') \) are isomorphic if there exists a bijection \( f : V \rightarrow V' \), such that

i) \( (f(u), f(v)) \in E' \) for all \( (u, v) \in E \)

\( (f^{-1}(u'), f^{-1}(v')) \in E \) for all \( (u', v') \in E' \)

ii) \( x'_{f(v)} = x_v \) for all \( v \in V \)

- In unlabelled case we can omit labels, or set \( x_v = 0 \) for all vertices \( v \in V \)
- Unknown whether it is solvable in polynomial time
WL Isomorphism Test

Overview

- Algorithm for solving the graph isomorphism problem
- Idea: Iteratively reduce graphs to canonical forms that coincide if graphs are isomorphic
- If canonical forms differ, graphs are non-isomorphic
- In each step $t$, assign to every node $i$ a label $x_i^t$

Figure 5: The Weisfeiler-Lehman Isomorphism Algorithm
**WL Isomorphism Test**

Algorithm\(^3\)

- Initialization: Set node features \( x_v^0 \) to original graph labels

---

*Figure 5: The Weisfeiler-Lehman Isomorphism Algorithm*
WL Isomorphism Test
Algorithm$^3$

- Initialization: Set node features $x_v^0$ to original graph labels
- For $t = 0,\ldots, n - 1$, repeat
  - For each node $v$ form a multi set $S_v^t$ of the labels of all neighbors
    \[ S_v^t = \{ x_u^t \mid u \in \mathcal{N}(v) \} \]
WL Isomorphism Test

Algorithm\textsuperscript{[3]}

- Initialization: Set node features $x_v^0$ to original graph labels

- For $t = 0, \ldots, n - 1$, repeat
  - For each node $v$ form a multi set $S^t_v$ of the labels of all neighbors
    \[ S^t_v = \{ x_u^t | u \in \mathcal{N}(v) \} \]
  - Map each pair of label $x_v^t$ and multi set $S^t_v$ to a new label $x_v^{t+1}$ (e.g. via a hash function)

Figure 5: The Weisfeiler-Lehman Isomorphism Algorithm
WL Isomorphism Test
Algorithm[3]

• Initialization: Set node features $x_v^0$ to original graph labels

• For $t = 0, \ldots, n - 1$, repeat
  • For each node $v$ form a multi set $S_v^t$ of the labels of all neighbors
    $S_v^t = \{x_u^t | u \in \mathcal{N}(v)\}$
  • Map each pair of label $x_v^t$ and multi set $S_v^t$ to a new label $x_v^{t+1}$ (e.g. via a hash function)

Figure 5: The Weisfeiler-Lehman Isomorphism Algorithm
WL Isomorphism Test
Algorithm\textsuperscript{[3]}

- Initialization: Set node features $x_v^0$ to original graph labels

- For $t = 0, \ldots, n - 1$, repeat
  - For each node $v$ form a multi set $S_v^t$ of the labels of all neighbors
    \[ S_v^t = \{ x_u^t \mid u \in \mathcal{N}(v) \} \]
  - Map each pair of label $x_v^t$ and multi set $S_v^t$ to a new label $x_v^{t+1}$ (e.g. via a hash function)

Figure 5: The Weisfeiler-Lehman Isomorphism Algorithm
WL Isomorphism Test
Algorithm[3]

- Initialization: Set node features $x^0_v$ to original graph labels

- For $t = 0, \ldots, n - 1$, repeat

  - For each node $v$ form a multi set $S^t_v$ of the labels of all neighbors
    \[ S^t_v = \{ x^t_u | u \in \mathcal{N}(v) \} \]

  - Map each pair of label $x^t_v$ and multi set $S^t_v$ to a new label $x^{t+1}_v$ (e.g. via a hash function)

Figure 5: The Weisfeiler-Lehman Isomorphism Algorithm
WL Isomorphism Test

Algorithm[^3]

- Initialization: Set node features $x_v^0$ to original graph labels

- For $t = 0, \ldots, n - 1$, repeat
  - For each node $v$ form a multi set $S^t_v$ of the labels of all neighbors
    \[ S^t_v = \{ x_u^t \mid u \in \mathcal{N}(v) \} \]
  - Map each pair of label $x_v^t$ and multi set $S^t_v$ to a new label $x_v^{t+1}$ (e.g. via a hash function)
  - Terminate if assignments of nodes to labels did not change from previous iteration

---

Figure 5: The Weisfeiler-Lehman Isomorphism Algorithm
WL Isomorphism Test

Algorithm\[^3\]

- Initialization: Set node features $x_v^0$ to original graph labels

- For $t = 0, \ldots, n - 1$, repeat
  - For each node $v$ form a multi set $S_v^t$ of the labels of all neighbors
    \[
    S_v^t = \{ x_u^t \mid u \in \mathcal{N}(v) \}
    \]
  - Map each pair of label $x_v^t$ and multi set $S_v^t$ to a new label $x_v^{t+1}$ (e.g. via a hash function)
  - Terminate if assignments of nodes to labels did not change from previous iteration

Graph 1
\[
\{6,6,7,8,8\}
\]

Graph 2
\[
\{6,6,7,8,8\}
\]
WL Isomorphism Test
Connection to GNNs

- Power of multi set function used in every layer of anonymous GNNs determines power in classifying graph isomorphism

**Theorem 1**

Every GNN is at most as powerful as the WL isomorphism test. \[^{[1]}\]

**Theorem 2**

A GNN is as powerful as the WL isomorphism test if its layer aggregate, combine and readout functions are injective. \[^{[1]}\]
GIN Model

Definition\textsuperscript{[1]}

- GNN model that is as powerful as the WL isomorphism test
- Node update in each layer defined via

\[
x_v^k = MLP^k \left( (1 + \epsilon^k)x_v^{k-1} + \sum_{u \in N(v)} x_u^{k-1} \right)
\]

where $\epsilon^k$ are learnable parameters and $MLP^k$ are learnable multi layer perceptrons
Figure 6: Performance of different GNN models on a selection of graph classification tasks\textsuperscript{[1]}

\[1\]
Less Powerful Models
One-layer perceptron\textsuperscript{[1]}

Lemma

There exist finite multi sets $X_1 \neq X_2$, such that for any linear mapping $W$

\[
\sum_{x \in X_1} ReLU(Wx) = \sum_{x \in X_2} ReLU(Wx)
\]

- Linear model (without bias term) fails to distinguish between some multi sets
- One-layer perceptron is not a universal approximator of multi set functions (unlike MLP)
## Less Powerful Models

Different Aggregation Schemes\(^1\)

<table>
<thead>
<tr>
<th>Aggregation Function</th>
<th>Sum</th>
<th>Mean</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Classification level</strong></td>
<td><strong>Multiset</strong></td>
<td><strong>Distribution</strong></td>
<td><strong>Set</strong></td>
</tr>
<tr>
<td><strong>Sample Input</strong></td>
<td><img src="image1.png" alt="Sample Input" /></td>
<td><img src="image2.png" alt="Sample Input" /></td>
<td><img src="image3.png" alt="Sample Input" /></td>
</tr>
<tr>
<td><strong>Failure Example</strong></td>
<td><img src="image4.png" alt="Failure Example" /></td>
<td><img src="image5.png" alt="Failure Example" /></td>
<td><img src="image6.png" alt="Failure Example" /></td>
</tr>
</tbody>
</table>
GNNs with port numbering
Non-anonymous GNNs

- Anonymous GNNs cannot distinguish between messages from different neighbors and are at most as powerful as the WL-test
- Idea: Assign port numbering to distinguish between different neighbours
Port Numbering

Definition: Port

A port of a graph $G$ is a pair $(v, i)$ where $v \in V$ and $i \in \{1, 2, \ldots, \deg(i)\}$. We denote the set of all ports of $G$ with $P(G)$.

Figure 7: Example of a consistent port numbering$^{[4]}$
Port Numbering

**Definition: Port**

A port of a graph $G$ is a pair $(v, i)$ where $v \in V$ and $i \in \{1,2,\ldots,\deg(i)\}$. We denote the set of all ports of $G$ with $P(G)$.

**Definition: Port Numbering**

A port numbering is a function $p : P(G) \rightarrow P(G)$, such that for any edge $(u, v)$, there exist $i, j$ with $p(u, i) = (v, j)$.

We call $p$ consistent if it is self-inverse, i.e. $p(p(v, i)) = (v, i)$.

Figure 7: Example of a consistent port numbering\(^4\)
Vector-vector consistent GNNs

- Let \( p \) be a consistent port numbering and denote its two components by \( p_1, p_2 \), i.e.

\[
p(v, i) = (p_1(v, i), p_2(v, i))
\]

- Extend anonymous GNNs by including consistent port numbering in layer input

\[
x_{v}^{t+1} = f_\theta \left( x_{v}^{t}, \left( x_{p_1(v,1)}^{t}, p_2(v,1) \right), \left( x_{p_1(v,1)}^{t}, p_2(v,1) \right), \ldots \left( x_{p_1(v,\Delta)}^{t}, p_2(v,\Delta) \right) \right)
\]

- Port numbering can be computed beforehand in linear time
Vector-vector consistent GNNs

Example

\[ x_{\nu}^{t+1} = f_\theta \left( x_{\nu}^t, \left(x_{p_1(v,1)}^t, p_2(v,1)\right), \left(x_{p_1(v,1)}^t, p_2(v,1)\right), \ldots \left(x_{p_1(v,\Delta)}^t, p_2(v,\Delta)\right) \right) \]

\[ = f_\theta \left( x_{\nu}^t, \left(x_{v_2}^t, 1\right), \left(x_{v_3}^t, 1\right), \left(x_{v_1}^t, 2\right) \right) \]

\[ p(v,1) = (v_2,1) \]

\[ p(v,2) = (v_3,1) \]

\[ p(v,3) = (v_1,2) \]
Vector-vector consistent GNNs

CPNGNNs

- Authors of [3] introduce Consistent Port Numbering Graph Neural Networks (CPNGNNS)

\[
x_{v}^{t+1} = ReLU \left( W^t \cdot CONCAT \left( x_{v}^{t}, x_{p_1(v,1)}^{t}, p_2(v,1), x_{p_1(v,1)}^{t}, p_2(v,1), \ldots x_{p_1(v,\Delta)}^{t}, p_2(v, \Delta) \right) \right)
\]

\[
x_{v}^{final} = MLP \left( x_{v}^{T} \right) \quad \text{(in final layer)}
\]

- CPNGNNs (and VVC-GNNs) are strictly more powerful than regular GNNs

- Example: Finding single leaf problem
Vector-vector consistent GNNs

CPNGNNs

- Authors of [3] introduce Consistent Port Numbering Graph Neural Networks (CPNGNNS)

\[ x_{v}^{t+1} = ReLU \left( W^t \cdot CONCAT \left( x_{v}^{t}, x_{p1(v,1)}^{t}, p_2(v,1), x_{p1(v,1)}^{t}, p_2(v,1), \ldots x_{p1(v,\Delta)}^{t}, p_2(v,\Delta) \right) \right) \]

\[ x_{v}^{final} = MLP \left( x_{v}^{T} \right) \quad \text{(in final layer)} \]

- CPNGNNs (and VVC-GNNs) are strictly more powerful than regular GNNs

- Example: Finding single leaf problem
Vector-vector consistent GNNs
Single Leaf Problem\textsuperscript{[4]}

- Input: star graph
- Output: A single marked leaf node
- Basic GNNs fail since different leaf nodes cannot be distinguished and output coincides

Figure 8: Example instance of single leaf problem
Distributed Computing
GNNs with unique vertex IDs

- Strictly more powerful than other GNN classes
- Turing universal under certain conditions
- Problems arise during training since GNNs with unique vertex IDs do not generalise well
- Limitations for GNNs with unique vertex IDs also hold for other types of GNNs
LOCAL and CONGEST

1. Communication network
   • Represented by graph $G$
   • Each node represents a machine and communicates only with its neighbors

Figure 9: The LOCAL model of computation
LOCAL and CONGEST

- Distributed computing models with unique node IDs

1. Communication network
   - Represented by graph $G$
   - Each node represents a machine and communicates only with its neighbors

2. Synchronous computation
   - Computation performed in synchronous rounds where each round consists of two steps
     i) Propagate messages between neighbors
     ii) Perform arbitrarily powerful computation for each node

Figure 9: The LOCAL model of computation
LOCAL and CONGEST

- Distributed computing models with unique node IDs

1. Communication network
   - Represented by graph $G$
   - Each node represents a machine and communicates only with its neighbors

2. Synchronous computation
   - Computation performed in synchronous rounds where each round consists of two steps
     
     i) Propagate messages between neighbors
     ii) Perform arbitrarily powerful computation for each node

3. Message size
   - In CONGEST model: restricted in size to $b$ bits

Figure 9: The LOCAL model of computation
LOCAL and CONGEST
Connection to GNNs

Theorem

Message passing GNNs with unique vertex IDs and Turing complete aggregate and combine functions are equivalent to algorithms in the LOCAL model of computation.[5]

- Allows us to infer limits for the computational complexity of GNNs by leveraging results from the LOCAL model
- Similarly, we can infer limits for GNNs with limited width, using results from the CONGEST model
Requirements for Turing Universality

Message passing GNNs are Turing universal under the following conditions

i) Each node is uniquely identified

ii) The aggregate and combine functions are Turing complete

iii) The depth of the GNN is larger than the diameter of the input graph

iv) The width of the GNN is unbounded

• Note that universality in the case of graph level classification is trivial if the \textit{READOUT} function is Turing complete
Requirements for Turing Universality

Message passing GNNs are Turing universal under the following conditions

i) Each node is uniquely identified

ii) The aggregate and combine functions are Turing complete

iii) The depth of the GNN is larger than the diameter of the input graph

iv) The width of the GNN is unbounded

- Note that universality in the case of graph level classification is trivial if the READOUT function is Turing complete

Required for equivalence to LOCAL model

Required for Turing universality in LOCAL model
Limits from CONGEST model

Theorem

If a problem $P$ cannot be solved in less than $d$ rounds in CONGEST using messages of at most $b$ bits, then $P$ cannot be solved by a GNN of depth $d$ and width $w = \mathcal{O}(b/\log(n))$.\cite{5}

- Yields limits for the depth and width of a GNN, even for local problems

- Example: $k$-cycle classification for $k \geq 4$ requires depth

\[
\begin{align*}
    d &= \Omega\left(\sqrt{n}/(w \log n)\right) \quad \text{if } k \text{ even}, \\
    d &= \Omega\left(n/(w \log n)\right) \quad \text{if } k \text{ odd}.
\end{align*}
\]
Limits from CONGEST model

Experimental results

Figure 10: Performance of GNNs with different depth and width on the 4-cycle problem (determining whether a graph contains a 4-cycle).\textsuperscript{[5]}
Communication Capacity and Limitations
Communication capacity

- Computational power of GNN dependent on its depth and width: motivates generalising notion of communication complexity

- Assume that each node feature vector takes values in some finite alphabet $\mathcal{S}$ with $s = |\mathcal{S}|$ symbols

**Definition**

Let $g$ be a GNN and fix a graph $G = (V, E)$. For any two disjoint sets $V_1, V_2 \subseteq V$, the communication capacity $c_g$ of $g$ (with respect to $G, V_1, V_2$) is the maximum number of symbols that can be transmitted from $V_1$ to $V_2$ and vice versa.\[^6\]

Figure 11: Example of a graph partition
Communication capacity

- The communication capacity of a GNN with respect to the partition $V_1, V_2$ depends on

  i) Its width $w$ and its depth $d$

  ii) The size of messages passed in each layer

  iii) The size of its global state (if included)

  iv) The smallest cut separating the two subsets $V_1, V_2$
Communication complexity

- Two players with respective inputs $x_a, x_b$ attempt to compute a function $f(x_a, x_b)$

- A communication protocol $\pi$ determines the sequence of exchanged symbols between the players

- The number of exchanged symbols is denoted by $|\pi(x_a, x_b)|$

**Definition**

The communication complexity $c_f$ of $f$ corresponds to the minimum worst-case length of any protocol that computes $f$:

$$c_f = \min_{\pi} \max_{(x_a, x_b) \in X_a \times X_b} |\pi(x_a, x_b)|$$
Hardness of Graph Isomorphism Problem

- We can relate communication capacity and complexity to derive limitations of GNNs for graph isomorphism

- Idea: Consider two random graphs connected by a small amount of edges

- Results also hold in expectation for these specific sets of graphs

**Theorem**

Let $g$ be a GNN using a majority-voting or consensus based \textit{READOUT} function. To compute the isomorphism class of every graph/tree of $n$ nodes, it must be that $c_g = \Omega(n^2)/c_g = \Omega(n)$.\textsuperscript{[6]}
Hardness of Graph Isomorphism Problem

Empirical Results

Figure 14: Performance of GNNs with different communication capacity on the graph isomorphism problem for a sample set of general graphs (a) and a set of trees (b)[6].
Oversquashing
Oversquashing

**Definition**

The problem radius $r$ of a graph problem corresponds to its required range of interaction.

- GNN requires at least $K \geq r$ layers
- Size of receptive field of a node grows exponentially in the number of layers $K$

$$|\mathcal{N}_v^K| = \mathcal{O}(\exp(K))$$

- For fixed length feature vector $x_v^I$, this leads to an exponential bottleneck

---

Figure 15: The bottleneck in GNNs with many layers[7]
Oversquashing
Example Problem

• In the *NeighborsMatch* problem the goal is to predict the label of a node based on its degree

• Solution requires propagation of information from all labeled nodes to target node

• Leads to bottleneck that prevents fitting the training data perfectly

Figure 16: The NeighborsMatch problem. The correct output label for the depicted graph is C.[7]
Oversquashing

Empirical Results

Figure 17: Performance of different GNNs on the NeighborsMatch problem. Underfitting (caused by oversquashing) can be observed from a problem radius of $r = 4$.\cite{7}
Conclusion
What we covered

• Hierarchy of different GNN classes

• Anonymous GNNs are (at most) equivalent to the WL isomorphism test w.r.t. graph isomorphism

• Connection between GNNs and models of distributed computation

• Requirements of Turing completeness of GNNs with unique vertex IDs

• Limitations based on depth and width, and, more generally, communication capacity

• The problem of oversquashing in deep GNNs and problems with a large radius
References


