Stanford CS224W: GNNs and Algorithmic Reasoning

CS224W: Machine Learning with Graphs
Joshua Robinson, Stanford University
http://cs224w.stanford.edu
Announcements

- Colab 5 due **EOD Tuesday**
Stanford CS224W: GNNs and Algorithmic Reasoning

CS224W: Machine Learning with Graphs
Joshua Robinson, Stanford University
http://cs224w.stanford.edu
20th century saw unprecedented development of algorithms

- Sorting, shortest paths, graph search, routing
- Algorithmic paradigms such as greedy, divide-and-conquer, parallelism, recursion, deterministic vs non-deterministic

\[
\text{MERGE-SORT}(A, p, r) \\
1 \quad \text{if } p < r \\
2 \quad q = \lfloor (p + r)/2 \rfloor \\
3 \quad \text{MERGE-SORT}(A, p, q) \\
4 \quad \text{MERGE-SORT}(A, q + 1, r) \\
5 \quad \text{MERGE}(A, p, q, r)
\]
The study of algorithms and data structures are one of the most coveted areas of computer science.

All of computing is built on top of these fundamental algorithms:
- 100% including ML!

But so far this class has (mostly) treated GNNs as a “new” type of graph algorithm.
This class:

\[ f(\text{Input graph}) = \text{node embeddings} \]

How to learn mapping function \( f \)?

Connection to classical graph algorithms unclear
So far treated GNNs as a “new” type of graph algorithm.

But in reality, graph ML has deep connections to the theory of computer science.

Today:
- Ground development of GNNs in context of prior graph algorithms
  - Deep connections between “classical” algorithms and GNNs
- Use to inform neural networks architecture design
Plan for Today

- **Part 1**
  - An algorithm GNNs can run

- **Part 2**
  - Algorithmic structure of neural network architectures

- **Part 3**
  - What class of graph algorithms can GNNs simulate?

- **Part 4**
  - Algorithmic alignment: a principle for neural net design
Other Reading

- The work of Petar Veličković
  - Lectures at Cambridge, expository papers, tutorials etc.
  - Some of today’s material drawn from Petar’s lectures
Stanford CS224W: GNNs and Classical Algorithms
Graph Neural Networks

- GNNs defined by computation process
- I.e., how information is propagated across the graph to compute node embeddings
GNNs as graph algorithms

- We define “message passing” a computational process.

- Message passing defines a class of algorithms on graphs.

- But it is not clear what algorithm(s).

- A clue to get started: we have already seen one algorithm GNNs can express...
GNNs can execute the 1-WL isomorphism test

- Recall lecture 6: GNNs at most as expressive as the 1-WL isomorphism test
- GIN is exactly as expressive as 1-WL
- Argument: show that GIN is a neural version of 1-WL

Let’s recall the test...
Stanford CS224W: GNNs and the Weisfeiler-Lehman Isomorphism Test

CS224W: Machine Learning with Graphs
Joshua Robinson, Stanford University
http://cs224w.stanford.edu
- Simple test for testing if two graphs are the same:
  - Assign each node a “color”
  - Randomly hash neighbor colors until stable coloring obtained
  - Read out the final color histogram
- Declare two graphs:
  - Non-isomorphic if final color histograms differ
  - Test inconclusive otherwise (i.e., we do not know for sure that two graphs are isomorphic if the counts are the same)
GNNs and the 1-WL isomorphism test

- Running the test...

\( \phi = \text{HASH function} \) (i.e., injective function)

(diagrams thanks to Petar Veličković)
GNNs and the 1-WL isomorphism test

Running the test...

\[ \phi = \text{HASH function} \]
(i.e., injective function)

(diagrams thanks to Petar Veličković)
GNNs and the 1-WL isomorphism test

Running the test...

\( \phi = \text{HASH function} \) (i.e., injective function)

(diagrams thanks to Petar Veličković)
GNNs and the $1$-WL isomorphism test

- Running the test...

$\phi = \text{HASH function}$ (i.e., injective function)

(diagrams thanks to Petar Veličković)
GNNs and the 1-WL isomorphism test

- Running the test...

\[ \phi = \text{HASH function} \]
(i.e., injective function)

(diagrams thanks to Petar Veličković)
GNNs and the 1-WL isomorphism test

- Running the test...

\[ \phi = HASH \text{ function (i.e., injective function)} \]

(diagrams thanks to Petar Veličković)
GNNs and the 1-WL isomorphism test

- Test does fail to distinguish some graphs, e.g.,

![Graph 1](image1)
![Graph 2](image2)
GNNs and the 1-WL isomorphism test

- We have seen GIN is as expressive as the 1-WL test
  - i.e., Given $G_1, G_2$, the following are equivalent:
    - there exist parameters s.t. $\text{GIN}(G_1) \neq \text{GIN}(G_2)$
    - 1-WL distinguishes $G_1, G_2$
- GIN is a “neural version” of the 1-WL algorithm
  - Replaces HASH function with learnable MLP
We have seen GIN is as expressive as the 1-WL test
i.e., Given $G_1, G_2$, the following are equivalent:
- there exist parameters s.t. GIN$(G_1) \neq$ GIN$(G_2)$
- 1-WL distinguishes $G_1, G_2$

GIN is a “neural version” of the 1-WL algorithm
But this does not mean that 1-WL is the only graph algorithm GNNs can simulate
An untrained GNN (random MLP = random hash) is close to the 1-WL test
GNNs and the 1-WL isomorphism test

- We have seen GIN is as expressive as the 1-WL test
  - i.e., Given $G_1, G_2$, the following are equivalent:
    - there exist parameters s.t. $\text{GIN}(G_1) \neq \text{GIN}(G_2)$
    - 1-WL distinguishes $G_1, G_2$
- GIN is a “neural version” of the 1-WL algorithm
- But this does not mean that 1-WL is the only graph algorithm GNNs can simulate
  - An untrained GNN (random MLP = random hash) is close to the 1-WL test
- Today’s question: what other algorithms can (trained) GNNs simulate?
Plan for Today

- **Part 1**
  - An algorithm GNNs can run

- **Part 2**
  - Algorithmic structure of neural network architectures

- **Part 3**
  - What class of graph algorithms can GNNs simulate?

- **Part 4**
  - Algorithmic alignment: a principle for neural net design
Stanford CS224W: Algorithmic structure of neural networks
A neural network architecture defines a learnable computer program

**Eventual Aim:** identify a broad class of “classical” (graph) algorithms that GNNs can easily learn

- This is different from our previous study of expressive power
Key perspective switch:

In this lecture, we are **not focusing on expressive power** (as in lecture 6).

Instead we are focused on what tasks an architecture can **easily learn to solve**

- For today: **easily = sample efficient** (not too much training data)

Key intuition:

- MLPs easily learn smooth functions (e.g., linear, log, exp)
- MLPs bad at learning complex function (e.g., sums of smooth functions - i.e., for-loops)
Neural Networks as Algorithms

- **Approach:** define progressively more complex algorithmic problems, and corresponding neural net architectures capable of solving each
Problem 1 (feature extraction):

- **Input:** “flat” features $\mathbf{x} \in \mathbb{R}^n$ (e.g., color, size, position)
- **Output:** scalar value $y$ (e.g., is it round and yellow?)
Problem 1 (feature extraction):

- Input: “flat” features $\mathbf{x} \in \mathbb{R}^n$ (e.g., color, size, position)
- Output: scalar value $y$ (e.g., is it round and yellow?)
- No other prior knowledge (minimal assumptions)
Problem 1: feature extraction

- Problem 1 (task on one object):
  - **Input**: “flat” features $\mathbf{x} \in \mathbb{R}^n$ (e.g., color, size, position)
  - **Output**: scalar value $y$ (e.g., is it round and yellow?)
- No other prior knowledge (minimal assumptions)
- **Q**: What neural network choice suits this problem?
- **A**: MLPs (multilayer perceptrons)
  - Universal approximator
  - Makes no assumptions on input/output structure
What Can Neural Networks Reason about? Xu et al. ICLR 2020

Architectures and Problem Type

- MLP
  - task on one object
  - \( \sim \) feature extraction

Let's consider tasks on many objects...
Problem 2: Summary statistics

Problem 2 (summary statistics):

- **Input:** a set of objects \( \{x_i\} \), each with features containing their coordinate and color \( x_i = [x_i^{\text{color}}, x_i^{\text{coordinate}}] \)
Problem 2 (summary statistics):

- **Input**: a set of objects \{x_i\}, each with features containing their coordinate and color \(x_i = [x_i^{\text{color}}, x_i^{\text{coordinate}}]\)
- **Task Output**: some aggregate property of the set (e.g., largest x-coordinate)

(Answer: 5)
Problem 2: Summary statistics

- **Problem 2 (summary statistics):**
  - **Input:** a set of objects \( \{x_i\} \), each with features containing their coordinate and color \( x_i = [x_i^{\text{color}}, x_i^{\text{coordinate}}] \)
  - **Task Output:** some aggregate property of the set (e.g., largest x-coordinate)
  - \( y(\{x_i\}) = \max_i(x_i^{\text{coordinate}}) \)

\[
\text{out} = -\infty
\]

For \( i = 1, \ldots \)

if \( x_i^{\text{coordinate}} > \text{out} \):

\[
\text{out} = x_i^{\text{coordinate}}
\]

Return \( \text{out} \)

(Answer: 5)
Problem 2: Summary statistics

- MLP model: $\text{MLP}(x_1, \ldots, x_n)$
- **Not** well suited to this task
- To learn max (and min) MLP **has to learn to execute a for-loop**
- This is a complex operation, MLP needs lots of data to learn

$$y(\{x_i\}) = \max_i (x_i^{\text{coordinate}})$$  \(\text{(Answer: 5)}\)
Problem 2: Summary statistics

- New DeepSet model:
  - $\text{DeepSet}(\{x_i\}) = \text{MLP}_1(\sum_i \text{MLP}_2(x_i))$
- Well suited to this task
- Why?

$y(\{x_i\}) = \max_i (x_i^{\text{coordinate}})$ (Answer: 5)
Problem 2: Summary statistics

- New DeepSet model:
  - DeepSet($\{x_i\}$) = $\text{MLP}_1(\sum_i \text{MLP}_2(x_i))$
- Well suited to this task
- Why? Can approx. softmax, a simple approx. to max
  - $\max_i(x_i^{\text{coordinate}}) \approx \log \left( \sum_i e^{x_i^{\text{coordinate}}} \right)$ ($\text{MLP}_1$ learns log, $\text{MLP}_2$ learns exp)

$y(\{x_i\}) = \max_i(x_i^{\text{coordinate}})$

(Answer: 5)
Problem 2: Summary statistics

- New DeepSet model:
  - DeepSet(\{x_i\}) = MLP_1(\sum_i MLP_2(x_i))
- Well suited to this task
- **Why?** Can approx. softmax, a simple approx. to min/max
  - \(\max_i(x_i^{\text{coordinate}}) \approx \log \left(\sum_i e^{x_i^{\text{coordinate}}}\right)\) (MLP_1 learns log, MLP_2 learns exp)
- **Key point:**
  - Consequence: MLPs only must learn **simple functions** (log / exp)
  - This can be done easily, without needing much data
- MLP can provably also learn this. But must learn complex for-loop, which requires lots of training data
Architectures and Problem Type

- **MLP**
  - Task on one object
  - ~ feature extraction

- **DeepSet**
  - Task on many objects
  - ~ summary statistics
  - $y(\{x_i\}) = \max_i (x_i^{coordinate})$

Let's consider a harder task on many objects…
Problem 3: Relational argmax

- Problem 3 (relational argmax):
  - Input: a set of objects \( \{x_i\} \), each with features containing their coordinate and color \( x_i = [x_i^{\text{color}}, x_i^{\text{coordinate}}] \)

(Answer: red, purple)
Problem 3 (relational argmax):

- **Input:** a set of objects \( \{x_i\} \), each with features containing their coordinate and color \( x_i = [x_i^{\text{color}}, x_i^{\text{coordinate}}] \)
- **Task Output:** property of pairwise relation (e.g., what are the colors of the two furthest away objects?)

(Answer: red, purple)
Problem 3: Relational argmax

Problem 3 (relational argmax):

- **Input**: a set of objects \( \{x_i\} \), each with features containing their coordinate and color \( x_i = [x_i^{\text{color}}, x_i^{\text{coordinate}}] \)

- **Task Output**: property of pairwise relation (e.g., what are the colors of the two furthest away objects?)

\[
y(\{x_i\}) = (x_{i_1}^{\text{color}}, x_{i_2}^{\text{color}}) \\
\text{s.t. } i_1, i_2 = \arg\max_{i_1, i_2} ||x_i^{\text{coordinate}} - x_j^{\text{coordinate}}||
\]

(Answer: red, purple)
Problem 3: Relational argmax

- **DeepSet poorly suited to modelling pairwise relations**
  - **Recall:** \( \text{DeepSet}(\{x_i\}) = \text{MLP}_2(\sum_i \text{MLP}_1(x_i) ) \)
  - **Reason:**
    - task requires comparing pairs of objects – i.e., a for-loop
    - each object processed independently by MLP\(_1\)
  - **Consequence:** MLP\(_2\) has to learn complex for-loop (hard)
- \( \sum_i \text{MLP}_1(x_i) \) provably cannot learn pairwise relations

**Theorem:** Suppose \( g(x, y) = 0 \) if and only if \( x = y \). Then there is no \( f \) such that \( g(x, y) = f(x) + f(y) \)

\[
y(\{x_i\}) = (x_i^{\text{color}}, x_i^{\text{color}}) \quad \text{s.t.} \quad i_1, i_2 = \text{argmax}_{i_1, i_2} ||x_i^{\text{coordinate}} - x_j^{\text{coordinate}}||
\]

(Answer: red, purple)
**Problem 3: Relational argmax**

- GNN well suited to this task: for-loop is built in!
  - E.g., recall GIN update
  - For \( i = 1, \ldots, n \):
    - \( h_{i}^{l+1} = \text{MLP}_2 (\text{MLP}_1(h_i^l) + \sum_{j \in N(i)} \text{MLP}_1(h_j^l)) \)
  - Update of node embedding depends on other nodes
    - \( \text{MLP}_1 \) computes distance from \( i \) to \( j \)
    - \( \text{MLP}_2 \) identifies **which pair is best in** \( \{(i, j)\}_{j \in N(i)} \)

\[ y(\{x_i\}) = (x_i^{\text{color}}, x_i^{\text{color}}) \quad \text{s.t.} \quad i_1, i_2 = \arg\max_{i_1, i_2} |x_i^{\text{coordinate}} - x_j^{\text{coordinate}}| \]
What Can Neural Networks Reason about? Xu et al. ICLR 2020

Architectures and Problem Type

- **MLP**
  - Task on one object
  - ~ feature extraction

- **DeepSet**
  - Task on many objects
  - ~ summary statistics (max value difference)
  - \( y(\{x_i\}) = \max_i(x_i^{\text{coordinate}}) \)

- **GNN**
  - Task on many objects
  - ~ pairwise relations (relational argmax)
  - \( y(\{x_i\}) = (x_{i_1}^{\text{color}}, x_{i_2}^{\text{color}}) \) s. t. \( i_1, i_2 = \arg\max_{i_1, i_2} ||x_i^{\text{coordinate}} - x_j^{\text{coordinate}}|| \)

In each case, the neural net architecture “fits” the computations needed to compute the target... we will come back to this.
Results in practice

- Task 2: maximum value
  MLP fails due to inability to compute max

- Task 3: relational argmax
  - Both DeepSet and MLP fail

What Can Neural Networks Reason about? Xu et al. ICLR 2020
GNNs are good at solving tasks that require relating pairs of objects (nodes)
- MLPs/DeepSets cannot do this easily since they have to learn for-loop

“Relational argmax” is just one problem that GNN can solve...

What is the general class of algorithms GNNs can run?
Plan for Today

- **Part 1**
  - An algorithm GNNs can run

- **Part 2**
  - Algorithmic structure of neural network architectures

- **Part 3**
  - What class of graph algorithms can GNNs simulate?

- **Part 4**
  - Algorithmic alignment: a principle for neural net design
Stanford CS224W: Algorithmic Class of GNNs

CS224W: Machine Learning with Graphs
Joshua Robinson, Stanford University
http://cs224w.stanford.edu
Dynamic Programming

- Fundamental algorithmic paradigm
- One of the most influential algorithm classes in computer science (lecture 6 in MIT’s intro to Comp Sci)
- Works by recursively breaking a problem into smaller instances of the same problem type

Algorithms that use dynamic programming (cont.)
- Recursion solutions to lattice models for protein CRN binding
- Backward induction as a subroutine for forward dynamic programming
- Method of undetermined coefficients can be used to solve the Bellman equation in infinite horizon, discounted, finite-investor dynamic optimization problems
- Many string algorithms including longest common subsequence, longest increasing subsequence, longest common substring, Levenshtein distance (edit distance)
- Many algorithm problems on graphs can be solved efficiently for graphs of bounded treewidth or bounded clique-width by using dynamic programming on a tree decomposition of the graph.
- The Viterbi–Fulkerson–Kawaguti (VFK) algorithm which determines whether and how a given string can be generated by a given context-free grammar
- Knuth’s word-wrap algorithm that minimizes raggedness when word wrapping text
- The use of transportation tables and relaxation tables in computer science
- The Viterbi algorithm (used for hidden Markov models, and particularly in part of speech tagging)
- The Earley algorithm (a type of chart parser)
- The Backus–Nielsen algorithm and other algorithms used in bioinformatics, including sequence alignment, structural alignment, and structural similarity parallelism
- Floyd algorithm for shortest paths algorithm
- Optimizing the order for chain matrix multiplication
- Polynomial-time algorithms for the subset sum, knapsack and partition problems
- The dynamic programming algorithm for computing the global distance between two sequence
- The Silverman–Jain algorithm for relational database query optimization
- Dijkstra algorithm for evaluating Dijkstra’s algorithm
- Cleary–Heath–Luepscher method for solving the problem where genes of interest are interrupted
- The value iteration method for finding optimal decision processes
- Some graph image edge cutting selection methods such as the “infrared” selection tool in Photoshop
- Some methods for solving the Ising model optimization problems
- Some methods for solving the travelling salesmen problem, either exactly (in exponential time) or approximately (e.g., via the hillcascading
- Recursion least squares method
- Least squares in multi-information retrieval
- Adaptive coding technique for electronic medical records
- Stochastic algorithms for solving the correspondence problem used in stereo vision
- Some sorting (computer science) image rendering
- The Bellman–Ford algorithm for finding the shortest distance in a graph
- Some approximate solution methods for the closest pair problem
- Some approximate solution methods for the traveling salesman problem
Task 4 (shortest path):

- **Input**: a weighted graph and a chosen source node
- **Output**: all shortest paths out of source node (shortest path tree)
Dynamic Programming

- Task 4 (shortest path):
  - **Input**: a weighted graph and a chosen source node
  - **Output**: all shortest paths out of source node (shortest path tree)
- Algorithmic solution: Bellman-Ford

![Diagram of a graph with weights and Bellman-Ford algorithm code]

**Bellman-Ford algorithm**

```plaintext
for k = 1 ... |S| - 1:
  for u in S:
    d[k][u] = min_v d[k-1][v] + cost(v, u)
```
GNNs are Dynamic Programs

- Dynamic programming has very similar form to GNN

**Graph Neural Network**

for \( k = 1 \ldots \text{GNN iter} \):

for \( u \) in \( S \):

No need to learn for-loops

\[
h_u^{(k)} = \sum_v \text{MLP}(h_v^{(k-1)}, h_u^{(k-1)})
\]

**Bellman-Ford algorithm**

for \( k = 1 \ldots |S|-1 \):

for \( u \) in \( S \):

\[
d[k][u] = \min_v d[k-1][v] + \text{cost}(v, u)
\]

Learns a simple reasoning step
**GNNs are Dynamic Programs**

- Dynamic programming has very similar form to GNN
- Both have nested for-loops over:
  - Number of GNN layers / iterations of BF
  - Each node in graph

---

**Graph Neural Network**

\[
\text{for } k = 1 \ldots \text{GNN iter:} \\
\text{for } u \text{ in } S: \quad \text{No need to learn for-loops} \\
\quad h_u^{(k)} = \sum_v \text{MLP}(h_v^{(k-1)}, h_u^{(k-1)})
\]

---

**Bellman-Ford algorithm**

\[
\text{for } k = 1 \ldots |S| - 1: \\
\text{for } u \text{ in } S: \\
\quad d[k][u] = \min_v d[k-1][v] + \text{cost}(v, u)
\]

---

*Learns a simple reasoning step*
GNNs are Dynamic Programs

- Dynamic programming has very similar form to GNN
- Both have nested for-loops over:
  - Number of GNN layers / iterations of BF
  - Each node in graph
- GNN aggregation + MLP only needs to learn sum + min
- An MLP trying to learn a DP has to learn double-nested for loop – really hard to do!

**Graph Neural Network**

```
for k = 1 ... GNN iter:
  for u in S:  No need to learn for-loops
    h_u^{(k)} = \Sigma_v MLP(h_v^{(k-1)}, h_u^{(k-1)})
```

**Bellman-Ford algorithm**

```
for k = 1 ... |S| - 1:
  for u in S:
    d[k][u] = min_v d[k-1][v] + cost(v, u)
```

Learns a simple reasoning step
GNNs are Dynamic Programs

- There is an even better choice of GNN...
  - Choose \textit{min activation} to match DP
  - Then MLP only needs to learn \textit{linear function}!

\begin{align*}
h_u^{(k)} &= \sum_v \text{MLP}^{(k)}(h_v^{(k-1)}, h_u^{(k-1)}, w(v, u)) \\
\text{MLP has to learn non-linear steps} \quad \times \\
\text{MLP learns linear steps} \quad \checkmark
\end{align*}
GNNs are Dynamic Programs

- We expect GNNs to be good at solving tasks that can be solved with DP
  - E.g., shortest paths

- Does this actually happen?
Results in practice

- Task 2: maximum value
  MLP fails due to inability to compute max

- Task 3: relational argmax
  - Both DeepSet and MLP fail

- Task 4: shortest path
  (dynamic programming)
  - Task shortest path length up to 7
  - 7 layer GNN gets best performance
Conclusion

- **Goal:** understand what tasks GNNs are good at solving
  - We are **not** focusing on expressivity
  - Instead we are interested in how easy it is to learn the solution (e.g., how much data the model needs to see)

- **GNN message passing is a dynamic programing algorithm**

- Consequence: GNNs are a good choice of architecture for tasks that can be solved by a DP (e.g., finding shortest paths)
Part 1
   An algorithm GNNs can run

Part 2
   Algorithmic structure of neural network architectures

Part 3
   What class of graph algorithms can GNNs simulate?

Part 4
   Algorithmic alignment: a principle for neural net design
Stanford CS224W: Algorithmic Alignment

CS224W: Machine Learning with Graphs
Joshua Robinson, Stanford University
http://cs224w.stanford.edu
In the previous section we studied what type of tasks GNNs excel at solving

- **Key idea:** focus on the algorithm that solves the task
- If the neural net can express the algorithm easily, then it’s a good choice of architecture

- **How to formulate a general principle?**
Algorithmic Alignment

Given a target algorithm \( g = g_m \circ \cdots \circ g_1 \), a neural network architecture \( f = f_m \circ \cdots \circ f_1 \) if:

- \( g_i \) a simple function
- \( f_i \) can express \( g_i \)
- Each \( f_i \) has few learnable parameters (so can learn \( g_i \) easily)

If you remember any phrase from today, let it be algorithmic alignment – all of today's lecture can be understood with this idea.

About how a model expresses a target function, not if (i.e., expressive power). Recall that an MLP is a universal approximator.

Intuition: overall algorithm can be learned more easily by learning individual simple steps.
GNN is **algorithmically aligned** to dynamic programming (DP)

But algorithmic alignment is a **general principle** for designing neural network architectures

So we should be able to use it to design entirely new neural networks given a particular problem
Many successful examples of this in the literature

- Neural Shuffle-Exchange Networks (Freivalds et al., NeurIPS’19)
  - Linearithmic algorithms
- Neural Execution of Graph Algorithms (Veličković et al., ICLR’20)
  - Improved dynamic programming
- PrediNet (Shanahan et al., ICML’20)
  - Predicate Logic
- IterGNNs (Tang et al., NeurIPS’20)
  - Iterative algorithms
- Pointer Graph Networks (Veličković et al., NeurIPS’20)
  - Pointer-based data structures
- Persistent Message Passing (Strathmann et al., ICLR’21 SimDL)
  - Persistent data structures
Stanford CS224W: Applications of Algorithmic Alignment

CS224W: Machine Learning with Graphs
Joshua Robinson, Stanford University
http://cs224w.stanford.edu
Application 1: building a network to solve a new task
- The subset-sum problem (NP-hard)

Application 2: building neural networks that can generalize out-of-distribution
- The linear algorithmic alignment hypothesis
**Task:** given a set of numbers $S$, decide if there exists a subset that sums to $k$
Solving an NP-hard Task: Subset Sum

- **Task:** given a set of numbers $S$, decide if there exists a subset that sums to $k$

- Known to be NP-hard, no DP algorithm can solve this (so GNN not suitable)
Solving an NP-hard Task: Subset Sum

- **Exhaustive Search Algorithm for solving subset sum:**
  - Loop over all subsets $\tau \in S$ and check if sum is $k$

- **Clearly not polynomial time... but can it inspire a neural net architecture?**

![Subset Sum Illustration]

<table>
<thead>
<tr>
<th>10</th>
<th>0</th>
<th>5</th>
<th>8</th>
<th>6</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
</table>

**sum = 15**

5 + 8 + 2 = 15
Solving an NP-hard Task: Subset Sum

- **Exhaustive Search Algorithm for solving subset sum:**
  - Loop over all subsets $\tau \in S$ and check if sum is $k$

- **Clearly not polynomial time... but can it inspire a neural net architecture?**

- **Neural Exhaustive Search:**
  - **Given** $S = \{X_1, \ldots, X_n\}$,
  - $\text{NES}(S) = \text{MLP}(\max_{\tau \subseteq S} \text{LSTM}(X_1, \ldots, X_{|\tau|}: X_1, \ldots, X_{|\tau|} \in \tau))$
    - Algorithmically aligned to exhaustive search:
      - LSTM learns if the sum $X_1 + \ldots + X_{|\tau|} = k$ (simple function)
      - Max aggregation identifies best subset
      - MLP maps to true/false value
Solving an NP-hard Task: Subset Sum

- **Result in practice**
- **Random guessing gets 50% accuracy**

**Neural Exhaustive Search:**

- **Given** $S = \{X_1, \ldots, X_n\}$,
- **NES($S$) = MLP ($\max_{\tau \subseteq S} \text{LSTM}(X_1, \ldots, X_{|\tau|}; X_1, \ldots, X_{|\tau|} \in \tau)$**
  - Algorithmically aligned to exhaustive search:
    - LSTM learns if the sum $X_1 + \ldots + X_{|\tau|} = k$ (simple function)
    - Max aggregation identifies best subset
    - MLP maps to true/false value

![Bar chart showing accuracy comparison between NES, GNN6, GNN1, Deep Sets, and MLP.]
Designing New Neural Nets with Algorithmic Alignment

- **Application 1**: building a network to solve a new task
  - The subset-sum problem (NP-hard)

- **Application 2**: building neural networks that can generalize out-of-distribution
  - The linear algorithmic alignment hypothesis
We have argued that algorithmic alignment can help inspire architectures well suited to particular tasks.

- By well suited, we mean generalizes well using little training data.

But true AI requires something stronger than this...

- Also needs to “extrapolate” to instances that look very different from the training data.
Algorithmic Alignment and Extrapolation

- Extrapolation is also called out-of-distribution generalization

- Extrapolation is a holy grail of AI, necessary for systems to behave reliably in unforeseen future situations

- Can algorithmic alignment help with extrapolation?
  - Let’s start with a simple but important observation
How MLPs extrapolate

- **Observation:** ReLU MLPs extrapolate *linearly*
Observation: ReLU MLPs extrapolate linearly.

Can be proved that extrapolation is perfect for linear target functions.

But ReLU MLPs cannot generalize for non-linear target functions...

The need for linearity for MLP extrapolation suggests a hypothesis for GNN extrapolation...
Linear Algorithmic Alignment Hypothesis

Linear algorithmic alignment implies a neural network can extrapolate to unseen data.
The Linear Algorithmic Alignment Hypothesis

Linear algorithmic alignment implies a neural network can extrapolate to unseen data.

Linear Algorithmic Alignment Hypothesis

Given a target algorithm \( g = g_m \circ \cdots \circ g_1 \), a neural network architecture \( f = f_m \circ \cdots \circ f_1 \) linearly aligns if:

- \( f_i \) can express \( g_i \)
- \( f_i \) contains a combination of non-linearities and MLPs
- Each MLP in \( f_i \) only has to learn a linear map to perfectly fit \( g_i \)
How GNNs extrapolate

- Recall GNN for learning dynamic programs
- GNN aggregation function is key
  - Min aggregation is linearly algorithmically aligned
  - Sum aggregation is not
- Does linear algorithmic alignment lead to extrapolation?

How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks, Xu et al. ICLR 2021
How GNNs extrapolate

- Does linear algorithmic alignment lead to extrapolation?

Max degree and shortest paths are DP tasks

Yes!
Neural networks can be viewed as programs, or algorithms

Different neural network architectures are better suited to learning different algorithms

Graph neural networks are dynamic programs

Algorithmic alignment: make the computations steps of the neural net closely match the computational steps of the target algorithm

- Learn quicker, extrapolate better