Graph Representation Learning for Algorithmic Reasoning

Petar Veličković

DL4G@WWW2020
21 April 2020
Problem-solving approaches

Neural networks

Algorithms

Merge-Sort($A$, $p$, $r$)

1. if $p < r$
2. $q = \lceil(p + r)/2\rceil$
3. Merge-Sort($A$, $p$, $q$)
4. Merge-Sort($A$, $q + 1$, $r$)
5. Merge($A$, $p$, $q$, $r$)

Algorithm figures: Cormen, Leiserson, Rivest and Stein. Introduction to Algorithms.
Problem-solving approaches

Neural networks

- Operate on raw inputs
- Generalise on noisy conditions
- Models reusable across tasks
- Require big data
- Unreliable when extrapolating
- Lack of interpretability

Algorithms

- Trivially strongly generalise
- Compositional (subroutines)
- Guaranteed correctness
- Interpretable operations
- Inputs must match spec
- Not robust to task variations
Problem-solving approaches

Neural networks

- Operate on raw inputs
- Generalise on noisy conditions
- Models reusable across tasks
- Require big data
- Unreliable when extrapolating
- Lack of interpretability

Algorithms

- Trivially strongly generalise
- Compositional (subroutines)
- Guaranteed correctness
- Interpretable operations
- Inputs must match spec
- Not robust to task variations

Is it possible to get the best of both worlds?
Problem-solving approaches

Neural networks

- Operate on raw inputs
- Generalise on noisy conditions
- Models reusable across tasks
- Require big data
- Unreliable when extrapolating
- Lack of interpretability

Algorithms

- Trivially strongly generalise
- Compositional (subroutines)
- Guaranteed correctness
- Interpretable operations
- Inputs must match spec
- Not robust to task variations

Is it possible to get the best of both worlds?

This talk!
Neural Graph-Algorithmic Reasoning

- *Can* neural nets robustly *reason* like algorithms?

- Algorithms manipulate (un)ordered sets of objects, and their relations.
  - They operate over *graphs*.
  - Supervise *graph neural networks* on algorithm execution tasks!

- Call this approach *neural graph algorithm execution*.
Why?

- Benchmarking graph neural nets
- Strong generalisation
- Multi-task learning
- Algorithm discovery
Why?

- Benchmarking graph neural nets
- Strong generalisation
- Multi-task learning
- Algorithm discovery
Benchmarked GNNs

- Popular GNN benchmark datasets often unreliable

Pitfalls of Graph Neural Network Evaluation

Oleksandr Shchur; Maximilian Mumme; Aleksandar Bojchevski, Stephan Günnemann
Technical University of Munich, Germany
{shchur,mumme,a.bojchevski,guennemann}@in.tum.de

On Graph Classification Networks, Datasets and Baselines

Enxhell Luzhnica *; Ben Day *; Pietro Lio *
Benchmarking GNNs

- Popular GNN benchmark datasets often unreliable
  - Complexity not very high

### Simplifying Graph Convolutional Networks

<table>
<thead>
<tr>
<th>Our experiments:</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>81.4 ± 0.4</td>
<td>70.9 ± 0.5</td>
<td>79.0 ± 0.4</td>
</tr>
<tr>
<td>GAT</td>
<td>83.3 ± 0.7</td>
<td>72.6 ± 0.6</td>
<td>78.5 ± 0.3</td>
</tr>
<tr>
<td>FastGCN</td>
<td>79.8 ± 0.3</td>
<td>68.8 ± 0.6</td>
<td>77.4 ± 0.3</td>
</tr>
<tr>
<td>GIN</td>
<td>77.6 ± 1.1</td>
<td>66.1 ± 0.9</td>
<td>77.0 ± 1.2</td>
</tr>
<tr>
<td>LNet</td>
<td>80.2 ± 3.0</td>
<td>67.3 ± 0.5</td>
<td>78.3 ± 0.6</td>
</tr>
<tr>
<td>AdaLNet</td>
<td>81.9 ± 1.9</td>
<td>70.6 ± 0.8</td>
<td>77.8 ± 0.7</td>
</tr>
<tr>
<td>DGI</td>
<td>82.5 ± 0.7</td>
<td>71.6 ± 0.7</td>
<td>78.4 ± 0.7</td>
</tr>
<tr>
<td>SGC</td>
<td>81.0 ± 0.0</td>
<td>71.9 ± 0.1</td>
<td>78.9 ± 0.0</td>
</tr>
</tbody>
</table>

### Setting | Model | Test F1 |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>GaAN</td>
<td>96.4</td>
</tr>
<tr>
<td></td>
<td>SAGE-mean</td>
<td>95.0</td>
</tr>
<tr>
<td></td>
<td>SAGE-LSTM</td>
<td>95.4</td>
</tr>
<tr>
<td></td>
<td>SAGE-GCN</td>
<td>93.0</td>
</tr>
<tr>
<td></td>
<td>FastGCN</td>
<td>93.7</td>
</tr>
<tr>
<td></td>
<td>GCN</td>
<td>OOM</td>
</tr>
<tr>
<td>Unsupervised</td>
<td>SAGE-mean</td>
<td>89.7</td>
</tr>
<tr>
<td></td>
<td>SAGE-LSTM</td>
<td>90.7</td>
</tr>
<tr>
<td></td>
<td>SAGE-GCN</td>
<td>90.8</td>
</tr>
<tr>
<td></td>
<td>DGI</td>
<td>94.0</td>
</tr>
<tr>
<td>No Learning</td>
<td>Random-Init DGI</td>
<td>93.3</td>
</tr>
<tr>
<td></td>
<td>SGC</td>
<td>94.9</td>
</tr>
</tbody>
</table>

\[ \hat{Y}_{SGC} = \text{softmax}(S^KX\Theta) \]
Benchmarking GNNs

- Popular GNN benchmark datasets often unreliable
  - Complexity not very high

- Algorithms prove very favourable
  - Infinite data
  - Complex data manipulation
  - A clear hierarchy of models emerges!

<table>
<thead>
<tr>
<th>Models</th>
<th>Accuracy @ seq_len = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEE</td>
<td>100.00%</td>
</tr>
<tr>
<td>Modified transformer</td>
<td>99.14%</td>
</tr>
<tr>
<td>Vanilla transformer</td>
<td>97.66%</td>
</tr>
</tbody>
</table>
Benchmarking GNNs

- Popular GNN benchmark datasets often unreliable
  - Complexity not very high

- Algorithms prove very favourable
  - Infinite data
  - Complex data manipulation
  - A clear hierarchy of models emerges!

- A clearly specified generating function
  - No noise in the data
  - Enabling rigorous credit assignment
Benchmarking GNNs

● Popular GNN benchmark datasets often unreliable
  ○ Complexity not very high

● Algorithms prove very favourable
  ○ Infinite data
  ○ Complex data manipulation
  ○ A clear hierarchy of models emerges!

● A clearly specified generating function
  ○ No noise in the data
  ○ Enabling rigorous credit assignment

● The world is propped-up on polynomial-time algorithms
  ○ Applicable to NP-hard problems (see e.g. Joshi, Laurent and Bresson, NeurIPS’19 GRL)
Why?

- Benchmarking graph neural nets
- Strong generalisation
- Multi-task learning
- Algorithm discovery
Strong generalisation

- Learning an algorithm is not learning input-output mapping!

(Graves et al., 2014)
Strong generalisation

- Learning an algorithm is not learning input-output mapping!

- Imitating individual operations enables strong generalisation.
  - Consider how humans devise algorithms “by hand”.
  - Scales to much larger test graph sizes.

Table 1. Performance of different tasks on variable sizes of test examples (trained with examples of size 8)

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Sizes</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection sort</td>
<td></td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Merge sort</td>
<td></td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Shortest paths</td>
<td></td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00*</td>
</tr>
</tbody>
</table>
**Strong generalisation**

- Learning an *algorithm* is **not** learning input-output *mapping*!

- Imitating individual *operations* enables **strong** generalisation.
  - Consider how humans devise algorithms “by hand”.
  - Scales to much larger test graph sizes.

- **Grounds** the GNN in the underlying algorithmic reasoning
  - Deep learning is about learning representations
  - Learn representations of *manipulations*!
Why?

Benchmarking graph neural nets

Strong generalisation

Multi-task learning

Algorithm discovery
Multi-task learning

- Learning representations of **manipulations**
  - lots of potential for representational reuse.
  - Many algorithms share **subroutines**.

MST-PRIM ($G, w, s$)

1. for each $u \in G.V$
2. \hspace{1em} $u.key = \infty$
3. \hspace{1em} $u.\pi = \text{NIL}$
4. \hspace{1em} $s.key = 0$
5. \hspace{1em} $Q = G.V$
6. \hspace{1em} while $Q \neq \emptyset$
7. \hspace{2em} $u = \text{EXTRACT-MIN}(Q)$
8. \hspace{2em} for each $v \in G.Adj[u]$
9. \hspace{3em} if $v \in Q$ and $w(u, v) < v.key$
10. \hspace{4em} $\text{DECREASE-KEY}(Q, v, w(u, v))$
11. \hspace{2em} $v.\pi = u$

DIJKSTRA ($G, w, s$)

1. for each $u \in G.V$
2. \hspace{1em} $u.key = \infty$
3. \hspace{1em} $u.\pi = \text{NIL}$
4. \hspace{1em} $s.key = 0$
5. \hspace{1em} $Q = G.V$
6. \hspace{1em} while $Q \neq \emptyset$
7. \hspace{2em} $u = \text{EXTRACT-MIN}(Q)$
8. \hspace{2em} for each $v \in G.Adj[u]$
9. \hspace{3em} if $u.key + w(u, v) < v.key$
10. \hspace{4em} $\text{DECREASE-KEY}(Q, v, u.key + w(u, v))$
11. \hspace{2em} $v.\pi = u$
Multi-task learning

- Learning representations of manipulations
  ⇒ lots of potential for representational reuse.
  ○ Many algorithms share subroutines.

- Representations can positively reinforce one another!
  ○ Meta-representation of algorithms.
  ○ Plentiful opportunity for:
    ■ Multi-task learning
    ■ Meta-learning
    ■ Continual learning
  with clearly defined task relations!
Multi-task learning

- Learning representations of **manipulations**
  - lots of potential for representational *reuse*.
  - Many algorithms share **subroutines**.

- Representations can positively **reinforce** one another!
  - **Meta-representation** of algorithms.
  - Plentiful opportunity for:
    - **Multi-task** learning
    - **Meta**-learning
    - **Continual** learning
      with clearly defined task relations!

- Output of **easier** algorithm can be used as *input* for a harder one.
Why?

- Benchmarking graph neural nets
- Strong generalisation
- Multi-task learning
- Algorithm discovery
Algorithm discovery

- Inspecting intermediate outputs of an algorithm can **decode** its behaviour!

- Opportunity for deriving **novel** algorithms, e.g.
  - Improved heuristics for *intractable* problems.
  - Optimising for GNN executors (e.g. GPU/TPU).
Algorithm discovery

- Inspecting intermediate outputs of an algorithm can **decode** its behaviour!

- Opportunity for deriving **novel** algorithms, e.g.
  - Improved heuristics for *intractable* problems.
  - Optimising for GNN executors (e.g. GPU/TPU).

- Machine learning ← **Competitive programming**!
  - My way into computer science :)
Algorithm discovery

- Inspecting intermediate outputs of an algorithm can **decode** its behaviour!

- Opportunity for deriving **novel** algorithms, e.g.
  - Improved heuristics for **intractable** problems.
  - Optimising for GNN executors (e.g. GPU/TPU).

- Machine learning ← **Competitive programming**!
  - My way into computer science :)

- **Conjecture:** Can perform **soft subroutine reuse** from polynomial-time algorithms.
Programming language hierarchy

- High level
- Middle level
- Low level
GNN-Algorithmic hierarchy

**WHAT CAN NEURAL NETWORKS REASON ABOUT?**

(Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)

**NEURAL EXECUTION OF GRAPH ALGORITHMS**

(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)

**NEURAL EXECUTION ENGINES**

(Yan, Swersky, Koutra, Ranganathan and Hashemi. 2020)
### GNN-Algorithmic hierarchy

(Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)

- Learns an **algorithm** end-to-end only
- Strong theoretical link between **generalisation power** and **algorithmic alignment**
- GNNs align well with **dynamic programming**!

(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)

- Supervises on atomic **steps** of an algorithm
- Out-of-distribution testing of various GNNs
- **Multi-task learning** + maximisation aggregators generalise stronger!

(Yan, Swersky, Koutra, Ranganathan and Hashemi. 2020)

- Learns to execute tiny **operations**, then composes them
- Binary encoding and conditional masking
- Achieves **perfect** strong generalisation!
GNN-Algorithmic hierarchy

What can Neural Networks Reason About?

Neural Execution of Graph Algorithms

Neural Execution Engines
What Can Neural Networks Reason About?

- Which networks are best suited for certain types of reasoning?
  - **Theorem**: better *structural alignment* implies better *generalisation*!
  - GNNs ~ dynamic programming

**Graph Neural Network**

```
for k = 1 ... GNN iter:
  for u in S:
    $h_u^{(k)} = \Sigma_v \text{MLP}(h_v^{(k-1)}, h_u^{(k-1)})$
```

Leads to a simple reasoning step

**Bellman-Ford algorithm**

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

Answer$[k][i] = \text{DP-Update}(\{\text{Answer}[k-1][j], j = 1 \ldots n\})$
Architectures under study

**MLPs**

~ feature extraction

**Deep Sets** (Zaheer et al., NeurIPS 2017)

~ summary statistics

**GNNs**

~ (pairwise) relations

\[
y = \text{MLP}(\|_{s \in S} X_s)
\]

\[
y = \text{MLP}_2 \left( \sum_{s \in S} \text{MLP}_1(X_s) \right)
\]

\[
h_s^{(k)} = \sum_{t \in S} \text{MLP}_1^{(k)} \left( h_s^{(k-1)}, h_t^{(k-1)} \right)
\]

\[
y = \text{MLP}_2 \left( \sum_{s \in S} h_s^{(K)} \right)
\]

(Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)
Empirical results

**Summary statistics**
What is the maximum value difference among treasures?

**Relational argmax**
What are the colors of the furthest pair of objects?

**Dynamic programming**
What is the cost to defeat monster X by following the optimal path?

(Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)
GNN-Algorithmic hierarchy

What Can Neural Networks Reason About?

Neural Execution of Graph Algorithms

Neural Execution Engines

(Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)

(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)

(Yan, Swersky, Koutra, Ranganathan and Hashemi. 2020)
Neural Execution of Graph Algorithms

Supervise on appropriate output values at every step.

Bellman–Ford algorithm

Message–passing neural network
Components of the executor

- **Encoder network**
  \[ z_i^{(t)} = f_A(x_i^{(t)}, h_i^{(t-1)}) \]

- **Processor network**
  \[ H^{(t)} = P(Z^{(t)}, E^{(t)}) \]

- **Decoder network**
  \[ \hat{y}_i^{(t)} = g_A(z_i^{(t)}, \overline{h}_i^{(t)}) \]

- **Termination network**
  \[ \tau^{(t)} = \sigma(T_A(H^{(t)})) \]

- **Repeat** as long as \( \tau^{(t)} > 0.5 \)

*algorithm-specific

- Hypothesis: **MPNN-max** is a highly suitable processor
Evaluation

- Evaluate on *parallel* and *sequential* algorithms.
  - Parallel: *Reachability* (**BFS**), *Shortest paths* (**Bellman-Ford**)
  - Sequential: *Minimal spanning trees* (**Prim**)
  - Explicit inductive bias on sequentiality (learnable mask!)

- Generate **graphs** from a wide variety of distributions:
  - Ladder, Grid, Tree, 4-Caveman, 4-Community, Erdős–Rényi, Barabási–Albert
  - Attach random-valued weights to each edge

- Study the “human-programmer” perspective: test generalisation from small graphs (20 nodes) to larger graphs (50/100 nodes).

- Learn to execute BFS and Bellman-Ford with **same** processor!
## Evaluation: Shortest paths (+ Reachability)

<table>
<thead>
<tr>
<th>Model</th>
<th>Predecessor (mean step accuracy / last-step accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20 nodes</td>
</tr>
<tr>
<td>LSTM (Hochreiter &amp; Schmidhuber, 1997)</td>
<td>47.20% / 47.04%</td>
</tr>
<tr>
<td>GAT* (Veličković et al., 2018)</td>
<td>64.77% / 60.37%</td>
</tr>
<tr>
<td>GAT-full* (Vaswani et al., 2017)</td>
<td>67.31% / 63.99%</td>
</tr>
<tr>
<td>MPNN-mean (Gilmer et al., 2017)</td>
<td>93.83% / 93.20%</td>
</tr>
<tr>
<td>MPNN-sum (Gilmer et al., 2017)</td>
<td>82.46% / 80.49%</td>
</tr>
<tr>
<td>MPNN-max (Gilmer et al., 2017)</td>
<td><strong>97.13% / 96.84%</strong></td>
</tr>
<tr>
<td>MPNN-max (curriculum)</td>
<td>95.88% / 95.54%</td>
</tr>
<tr>
<td>MPNN-max (no-reach)</td>
<td>82.40% / 78.29%</td>
</tr>
<tr>
<td>MPNN-max (no-algo)</td>
<td>78.97% / 95.56%</td>
</tr>
</tbody>
</table>

Trained on 20-node graphs!

**Trained without reachability objective**

**Trained without step-wise supervision**

(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)
## Evaluation: Sequential execution

The sequential inductive bias is very helpful!

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy (next MST node / MST predecessor)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20 nodes</td>
</tr>
<tr>
<td>LSTM (Hochreiter &amp; Schmidhuber, 1997)</td>
<td>11.29% / 52.81%</td>
</tr>
<tr>
<td>GAT* (Veličković et al., 2018)</td>
<td>27.94% / 61.74%</td>
</tr>
<tr>
<td>GAT-full* (Vaswani et al., 2017)</td>
<td>29.94% / 64.27%</td>
</tr>
<tr>
<td>MPNN-mean (Gilmer et al., 2017)</td>
<td><strong>90.56% / 93.63%</strong></td>
</tr>
<tr>
<td>MPNN-sum (Gilmer et al., 2017)</td>
<td>48.05% / 77.41%</td>
</tr>
<tr>
<td>MPNN-max (Gilmer et al., 2017)</td>
<td>87.85% / 93.23%</td>
</tr>
<tr>
<td>MPNN-max (no-algo)</td>
<td>— / 71.02%</td>
</tr>
</tbody>
</table>
GNN-Algorithmic hierarchy

What Can Neural Networks Reason About?

Neural Execution of Graph Algorithms

Neural Execution Engines

(Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)

(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)

(Yan, Swersky, Koutra, Ranganathan and Hashemi. 2020)
Neural Execution Engines

- Teach a neural net to strongly perform tiny tasks (e.g. sum, product, argmin)
  - Compose tasks to specify algorithms
  - The building blocks must stay robust with long/OOD rollouts!

- Key components:
  - Bitwise embeddings
  - Transformers
  - Conditional masking

(Yan, Swersky, Koutra, Raganathan and Hashemi. 2020)
Learning to **selection sort** by composing **argmin**

```python
selection_sort(data):
    sorted_list = []
    while (len(data) > 0):
        min_index, min_element = find_min(data)
        data.delete(min_index)
        sorted_list.append(min_element)
    return sorted_list

find_min(data):
    min_element = -1
    min_index = -1
    for index, element in enumerate(data):
        if (element < min_element):
            min_element = element
            min_index = index
    return [min_index, min_element]
```
Learning to **selection** sort by composing **argmin**

<table>
<thead>
<tr>
<th>Models</th>
<th>Accuracy @ seq_len = 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEE</td>
<td>100.00%</td>
</tr>
<tr>
<td>Modified transformer</td>
<td>99.14%</td>
</tr>
<tr>
<td>Vanilla transformer</td>
<td>97.66%</td>
</tr>
</tbody>
</table>

(Yan, Swersky, Koutra, Raganathan and Hashemi. 2020)
Composing subroutines (Dijkstra)

\[
\text{shortest\_path}(\text{graph}, \text{source\_node}, \text{shortest\_path}): \\
dists = [] \\
\text{nodes} = [] \\
\text{anchor\_node} = \text{source\_node} \\
\text{node\_list} = \text{graph}\_\text{get}\_\text{nodes}() \\
\text{while node\_list:} \\
\quad \text{possible\_paths} = \text{sum}(\text{graph}\_\text{adj}(\text{anchor\_node}), \\
\quad \quad \text{shortest\_path}(\text{anchor\_node})) \\
\quad \text{shortest\_path} = \text{min}(\text{possible\_paths}, \text{shortest\_path}) \\
\quad \text{anchor\_node, min\_dist} = \text{min}(\text{shortest\_path}) \\
\quad \text{node\_list}\_\text{delete}(\text{anchor\_node}) \\
\quad \text{nodes}\_\text{append}(\text{anchor\_node}) \\
\quad \text{dists}\_\text{append}(\text{min\_dist}) \\
\text{return dists, nodes}
\]
**Recursive subroutines (Merge sort)**

```python
merge_sort(data, start, end):
    if (start < end):
        mid = (start + end) / 2
        merge_sort(data, start, mid)
        merge_sort(data, mid+1, end)
        merge(data, start, mid, end)
```

*Table 1. Performance of different tasks on variable sizes of test examples (trained with examples of size 8)*

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Sizes</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection sort</td>
<td></td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Merge sort</td>
<td></td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
<tr>
<td>Shortest paths</td>
<td></td>
<td>100.00</td>
<td>100.00</td>
<td>100.00</td>
<td>100.00*</td>
</tr>
</tbody>
</table>

(Yan, Swersky, Koutra, Raganathan and Hashemi. 2020)
Conclusions

- **Algorithmic reasoning** is an exciting novel area for **graph representation learning**!
  - Three concurrent works explore it at different levels:
    - Algo-level \((Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)\)
    - Step-level \((Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)\)
    - Unit-level \((Yan, Swersky, Koutra, Raganathan and Hashemi. 2020)\)

- Many questions left to be answered, at all levels of the hierarchy!
  - \(<Your\ contribution\ here/>\)
Thank you!

Questions?

petarv@google.com | https://petar-v.com

In collaboration with Charles Blundell, Raia Hadsell, Rex Ying, Matilde Padovano, Lars Buesing, Matt Overlan, Razvan Pascanu and Oriol Vinyals