



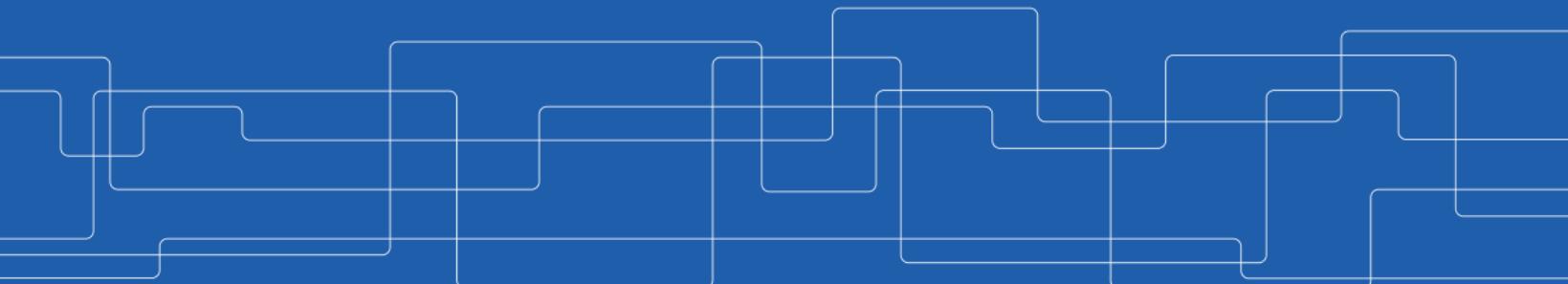
Distributed Learning - Model Parallelization

Model Parallelization

Amir H. Payberah

payberah@kth.se

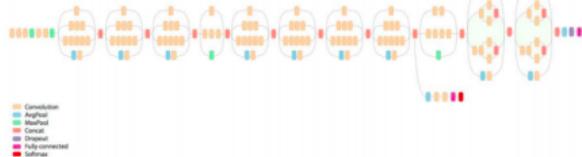
2020-10-26



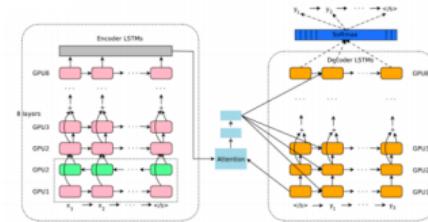


The Course Web Page

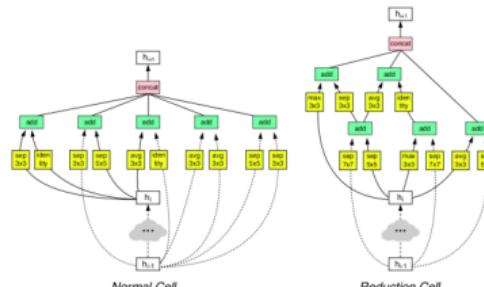
<https://fid3024.github.io>



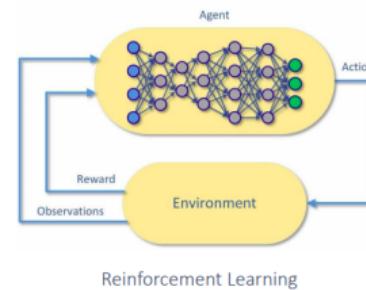
Convolutional Neural Networks



Recurrent Neural Networks



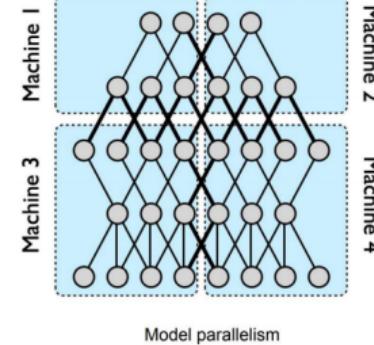
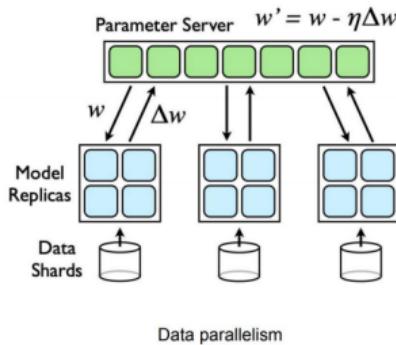
Neural Architecture Search



Reinforcement Learning

- ▶ Train **large deep learning models** with **huge amounts of training data**.
- ▶ Parallelization and distribution are essential.

Popular Parallelization Methods



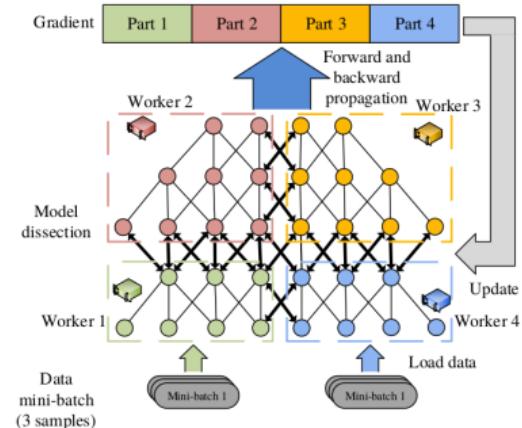
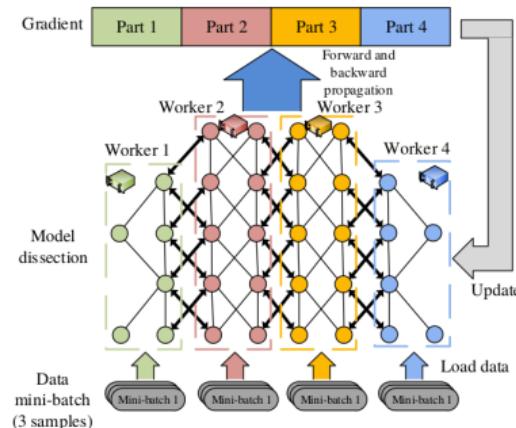
[Dean et al., Large Scale Distributed Deep Networks, 2012]



Model Parallelization

Model Parallelization

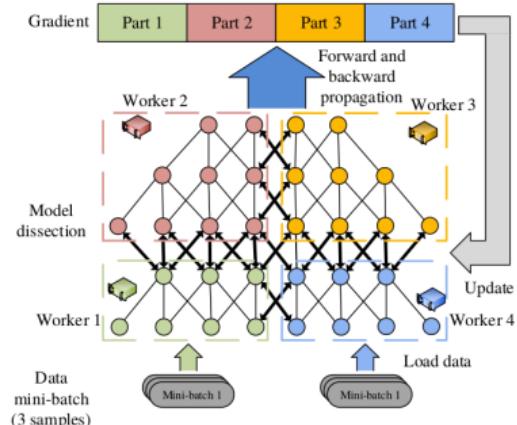
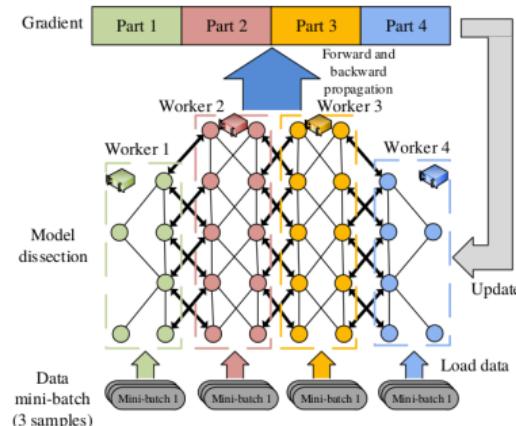
- The model is split across multiple devices.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

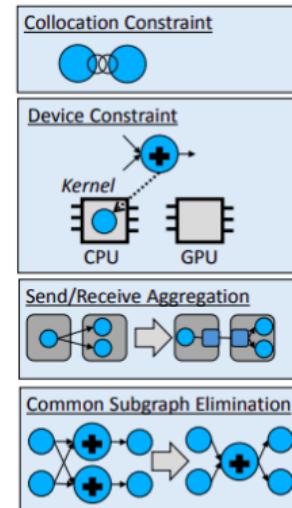
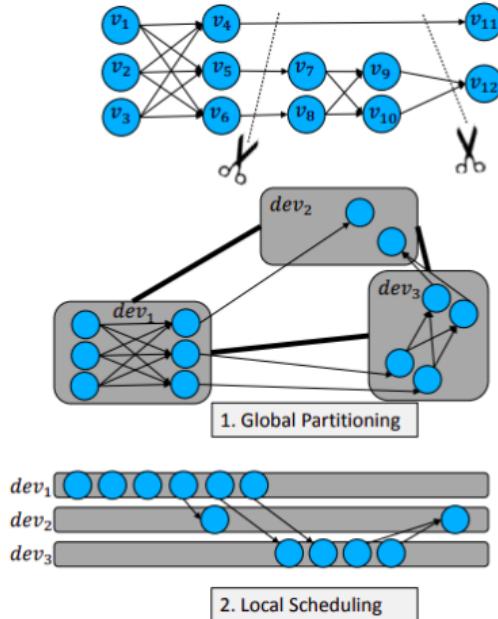
Model Parallelization

- The model is split across multiple devices.
- Depends on the architecture of the NN.



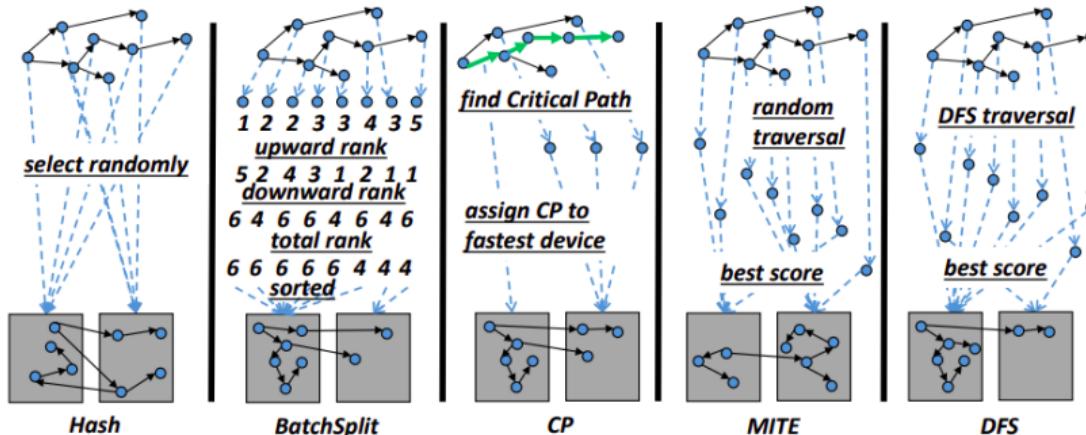
[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

NP-Completeness



[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]

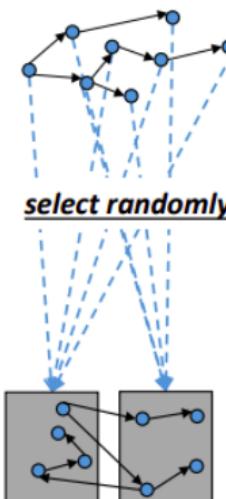
Partitioning Approaches



[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]

Model Parallelization - Hash Partitioning

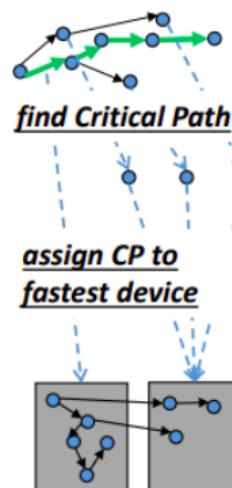
- ▶ Randomly assign vertices to devices proportionally to the capacity of the devices by using a [hash function](#).



[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]

Model Parallelization - Critical Path

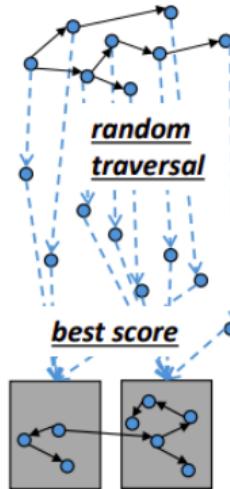
- ▶ Assigning the complete **critical path** to the fastest device.
- ▶ **Critical path**: the path with the **longest computation time** from source to sink vertex.



[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]

Model Parallelization - Multi-Objective Heuristics

- ▶ Different **objectives**, e.g., memory, importance, traffic, and execution time

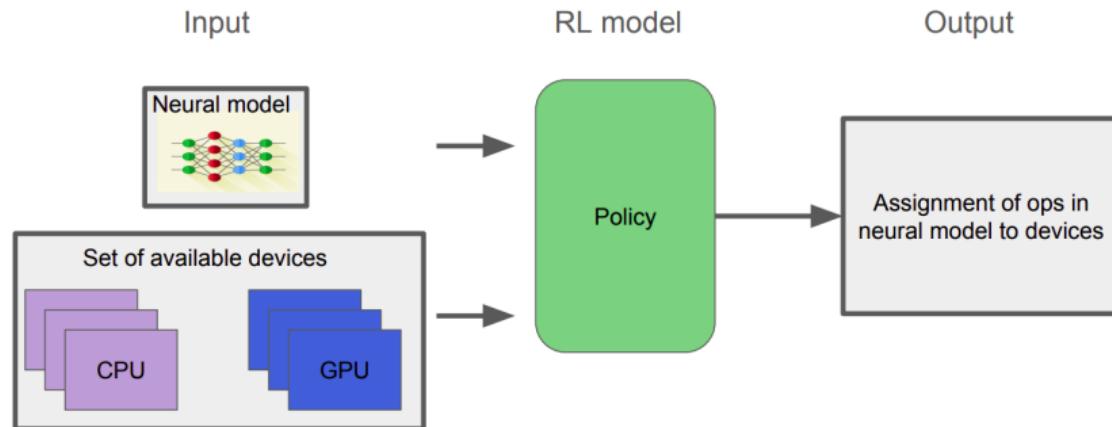


[Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017]



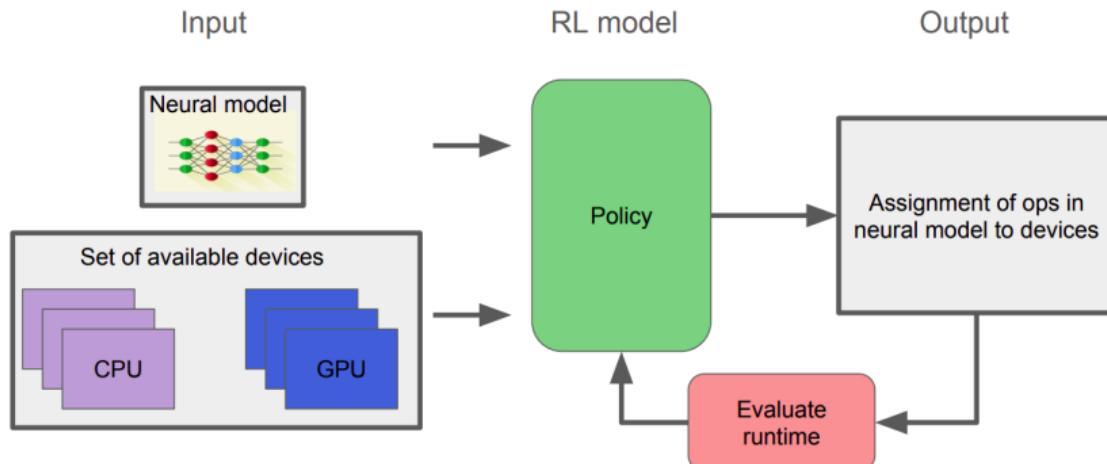
ML for Model Parallelization

Device Placement using Reinforcement Learning (1/3)



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

Device Placement using Reinforcement Learning (2/3)



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]



Device Placement using Reinforcement Learning (3/3)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$



Device Placement using Reinforcement Learning (3/3)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ $J(w)$: expected runtime

Device Placement using Reinforcement Learning (3/3)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ $J(w)$: expected runtime
- ▶ w : parameters of the RL policy



Device Placement using Reinforcement Learning (3/3)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ $J(w)$: expected runtime
- ▶ w : parameters of the RL policy
- ▶ \mathcal{G} : input neural graph

Device Placement using Reinforcement Learning (3/3)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ $J(w)$: expected runtime
- ▶ w : parameters of the RL policy
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime

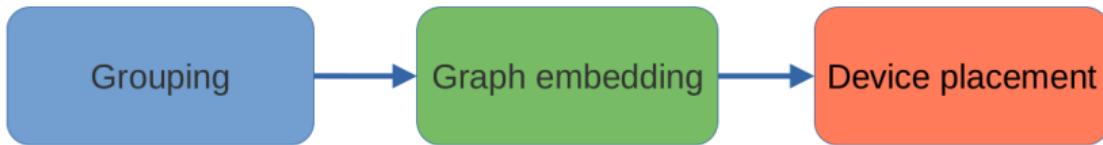
Device Placement using Reinforcement Learning (3/3)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ $J(w)$: expected runtime
- ▶ w : parameters of the RL policy
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime
- ▶ \mathcal{P} : output placements

Device Placement using Reinforcement Learning (3/3)

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)}[R(\mathcal{P})|\mathcal{G}]$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ $J(w)$: expected runtime
- ▶ w : parameters of the RL policy
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime
- ▶ \mathcal{P} : output placements
- ▶ $\pi(\mathcal{P}|\mathcal{G}, w)$: the RL policy (device placement policy)

Device Placement Policy





Solution 1

Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017
Mirhoseini et al., A Hierarchical Model for Device Placement, 2018

Device Placement Policy

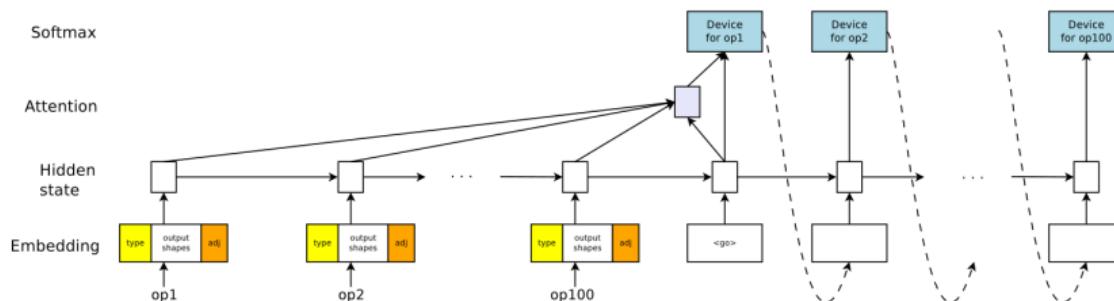


Device Placement Policy



System Overview

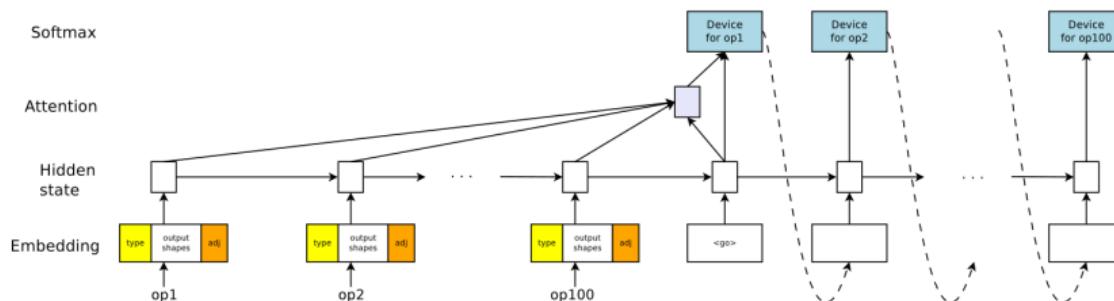
- ▶ The RL policy is defined as a attentional seq-to-seq model.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

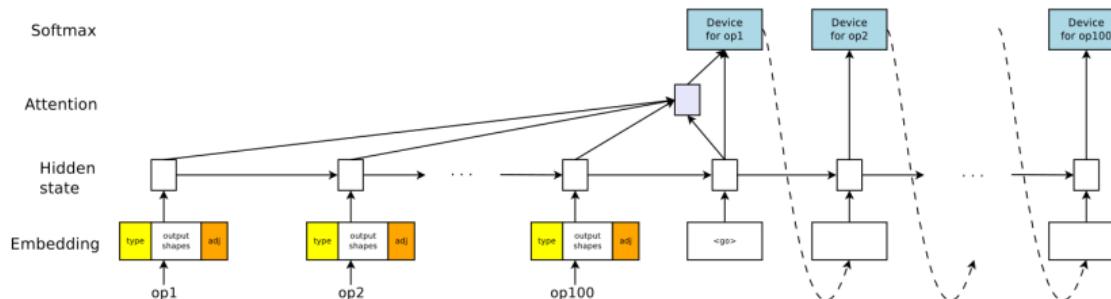
System Overview

- ▶ The RL policy is defined as a attentional seq-to-seq model.
- ▶ RNN Encoder receives sequence of embedding for each operation.



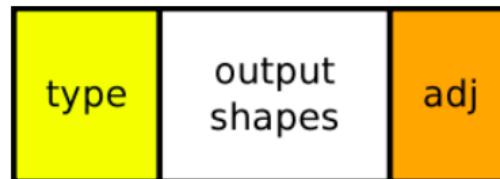
System Overview

- ▶ The RL policy is defined as a attentional seq-to-seq model.
- ▶ RNN Encoder receives sequence of embedding for each operation.
- ▶ RNN Decoder predicts a device placement for each operation.



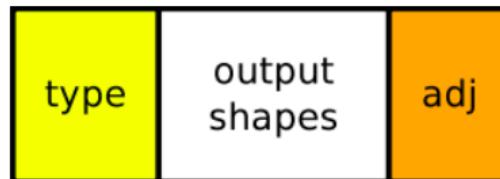
Operation Embedding

- ▶ The **embedding** of each operation is the **concatenation** of its **type**, its **output shape**, and its **one-hot encoded adjacency information**.



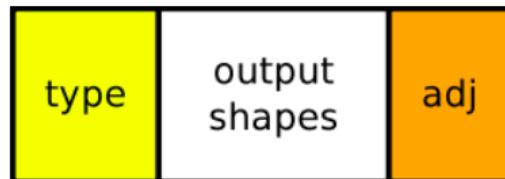
Operation Embedding

- ▶ The **embedding** of each operation is the **concatenation** of its **type**, its **output shape**, and its **one-hot encoded adjacency information**.
- ▶ Type of the operations, e.g., `MatMul` or `conv2d`.



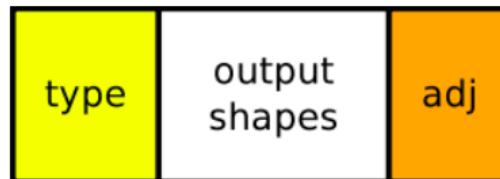
Operation Embedding

- ▶ The **embedding** of each operation is the **concatenation** of its **type**, its **output shape**, and its **one-hot encoded adjacency information**.
- ▶ Type of the operations, e.g., `MatMul` or `conv2d`.
- ▶ The size of each operation's list of output tensors (the **output shape**).



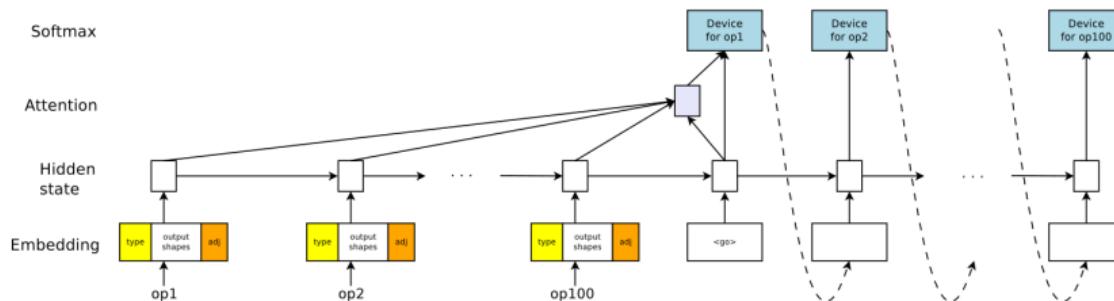
Operation Embedding

- ▶ The **embedding** of each operation is the **concatenation** of its **type**, its **output shape**, and its **one-hot encoded adjacency information**.
- ▶ Type of the operations, e.g., `MatMul` or `conv2d`.
- ▶ The size of each operation's list of output tensors (the **output shape**).
- ▶ The one-hot encoding vector that represents the operations that are direct inputs and outputs to each operation.



RNN Decoder

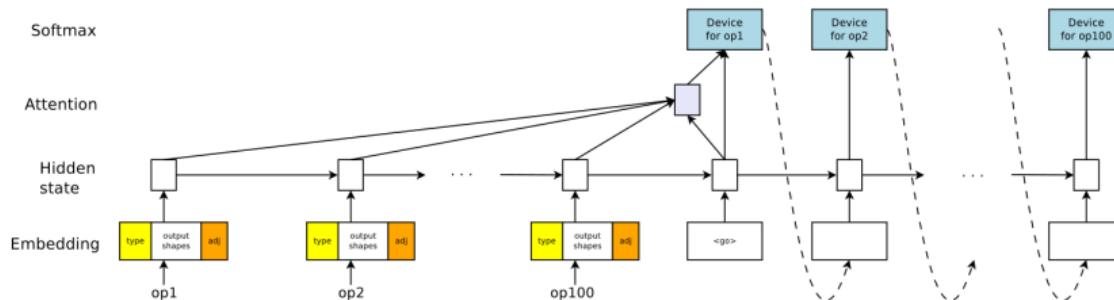
- ▶ The decoder is an **attentional LSTM** with a **fixed number of time steps**.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

RNN Decoder

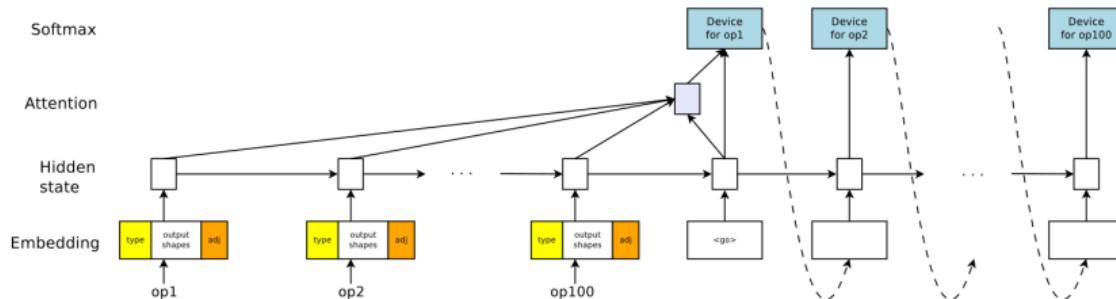
- ▶ The decoder is an **attentional LSTM** with a **fixed number of time steps**.
- ▶ The number of the steps is equal to the **number of operations** in a graph \mathcal{G} .



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]

RNN Decoder

- ▶ The decoder is an **attentional LSTM** with a **fixed number of time steps**.
- ▶ The number of the steps is equal to the **number of operations** in a graph \mathcal{G} .
- ▶ At each step, the decoder outputs the **device** for the **operation**.



[Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017]



Training with REINFORCE

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)} [R(\mathcal{P})|\mathcal{G}]$



Training with REINFORCE

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)} [R(\mathcal{P})|\mathcal{G}]$
- ▶ $\nabla_w J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)} [R(\mathcal{P}) \cdot \nabla_w \log_p(\mathcal{P}|\mathcal{G}, w)]$



Training with REINFORCE

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)} [R(\mathcal{P})|\mathcal{G}]$
- ▶ $\nabla_w J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)} [R(\mathcal{P}) \cdot \nabla_w \log_p(\mathcal{P}|\mathcal{G}, w)]$
- ▶ Estimate $\nabla_w J(w)$ by drawing K placement samples using $\mathcal{P} \sim \pi(\cdot|\mathcal{G}, w)$.



Training with REINFORCE

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)} [R(\mathcal{P})|\mathcal{G}]$
- ▶ $\nabla_w J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)} [R(\mathcal{P}) \cdot \nabla_w \log_p(\mathcal{P}|\mathcal{G}, w)]$
- ▶ Estimate $\nabla_w J(w)$ by drawing K placement samples using $\mathcal{P} \sim \pi(\cdot|\mathcal{G}, w)$.
- ▶ $\nabla_w J(w) = \frac{1}{K} \sum_{i=1}^K [R(\mathcal{P}_i - B) \cdot \nabla_w \log_p(\mathcal{P}|\mathcal{G}, w)]$



Training with REINFORCE

- ▶ $J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)} [R(\mathcal{P})|\mathcal{G}]$
- ▶ $\nabla_w J(w) = \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|\mathcal{G}, w)} [R(\mathcal{P}) \cdot \nabla_w \log_p(\mathcal{P}|\mathcal{G}, w)]$
- ▶ Estimate $\nabla_w J(w)$ by drawing K placement samples using $\mathcal{P} \sim \pi(\cdot|\mathcal{G}, w)$.
- ▶ $\nabla_w J(w) = \frac{1}{K} \sum_{i=1}^K [R(\mathcal{P}_i) - B] \cdot \nabla_w \log_p(\mathcal{P}|\mathcal{G}, w)$
- ▶ Estimate B : a baseline term to reduce the variance of the policy gradient.



Shortcomings of the Proposed Model

- ▶ Seq-to-seq models cannot be unrolled for more than few hundred steps.



Shortcomings of the Proposed Model

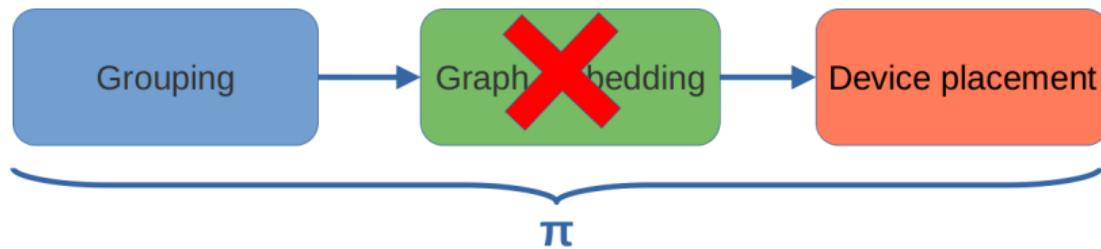
- ▶ Seq-to-seq models cannot be unrolled for more than few hundred steps.
- ▶ Most TensorFlow graphs contain tens of thousands of operations.



Shortcomings of the Proposed Model

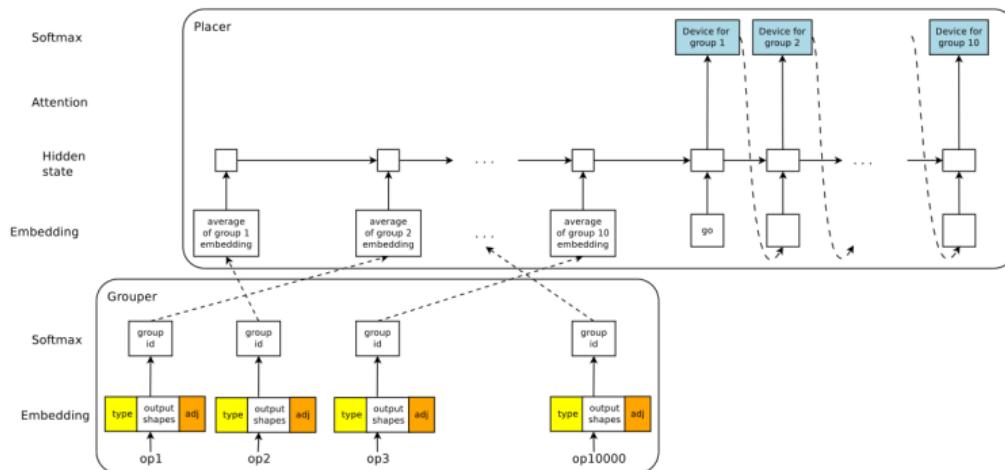
- ▶ Seq-to-seq models cannot be unrolled for more than few hundred steps.
- ▶ Most TensorFlow graphs contain tens of thousands of operations.
- ▶ Manual grouping of operations hampers scalability.

Device Placement Policy



An End-to-End Hierarchical Placement Model

- ▶ Grouping operations.
- ▶ Prediction is for **group placement**, not for a single operation.



[Mirhoseini et al., A Hierarchical Model for Device Placement, 2018]



Hierarchical Device Placement Optimization (1/2)

- ▶ $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d$
- ▶ Objective: $\arg \min_w J(w)$



Hierarchical Device Placement Optimization (1/2)

- ▶ $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph



Hierarchical Device Placement Optimization (1/2)

- ▶ $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime

Hierarchical Device Placement Optimization (1/2)

- ▶ $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g) p(d|g, w_g) R_d$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime
- ▶ $J(w_g, w_d)$: expected runtime

Hierarchical Device Placement Optimization (1/2)

- ▶ $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime
- ▶ $J(w_g, w_d)$: expected runtime
- ▶ w_g : parameters of the grouper

Hierarchical Device Placement Optimization (1/2)

- ▶ $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g) p(d|g, w_g) R_d$
- ▶ Objective: $\arg \min_w J(w)$
- ▶ \mathcal{G} : input neural graph
- ▶ R : runtime
- ▶ $J(w_g, w_d)$: expected runtime
- ▶ w_g : parameters of the grouper
- ▶ w_d : parameters of the placer



Hierarchical Device Placement Optimization (2/2)

► $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g) p(d|g, w_g) R_d$

Hierarchical Device Placement Optimization (2/2)

- ▶ $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g) p(d|g, w_g) R_d$
- ▶ $p(g, w_g)$: the probability of a sample group assignment g drawn from the **Grouper softmax distribution** π_g .



Hierarchical Device Placement Optimization (2/2)

- ▶ $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g)p(d|g, w_g)R_d$
- ▶ $p(g, w_g)$: the probability of a sample group assignment g drawn from the **Grouper softmax distribution** π_g .
- ▶ $p(d|g, w_g)$: the probability of a sample device placement d drawn from the **Placer softmax distribution** π_d .



Training with REINFORCE

► $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d, w_g, w_d)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g) p(d|g, w_g) R_d$



Training with REINFORCE

- ▶ $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d|g, w_g)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g) p(d|g, w_g) R_d$
- ▶ $\nabla_{w_g} J(w_g, w_d) = \sum_{g \sim \pi_g} \nabla_{w_g} p(g, w_g) \sum_{d \sim \pi_d} p(d|g, w_g) R_d$



Training with REINFORCE

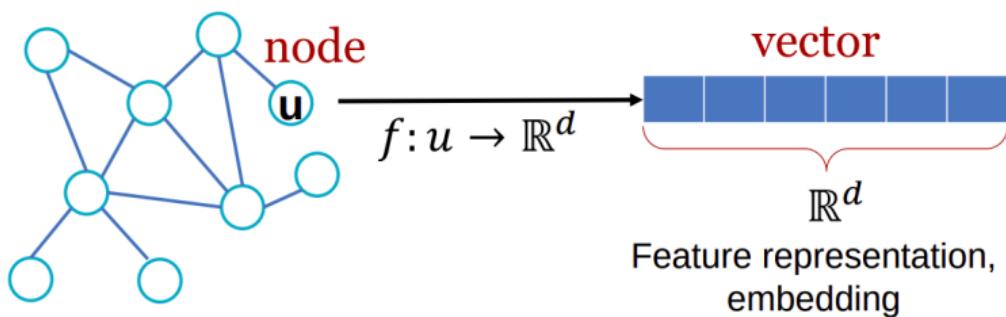
- ▶ $J(w_g, w_d) = \mathbb{E}_{\mathcal{P}(d|g, w_g)}[R_d] = \sum_{g \sim \pi_g} \sum_{d \sim \pi_d} p(g, w_g) p(d|g, w_g) R_d$
- ▶ $\nabla_{w_g} J(w_g, w_d) = \sum_{g \sim \pi_g} \nabla_{w_g} p(g, w_g) \sum_{d \sim \pi_d} p(d|g, w_g) R_d$
- ▶ $\nabla_{w_d} J(w_g, w_d) = \sum_{d \sim \pi_d} \sum_{g \sim \pi_g} p(g, w_g) \nabla_{w_d} p(d|g, w_g) R_d$



A Few Words About Graph Embedding

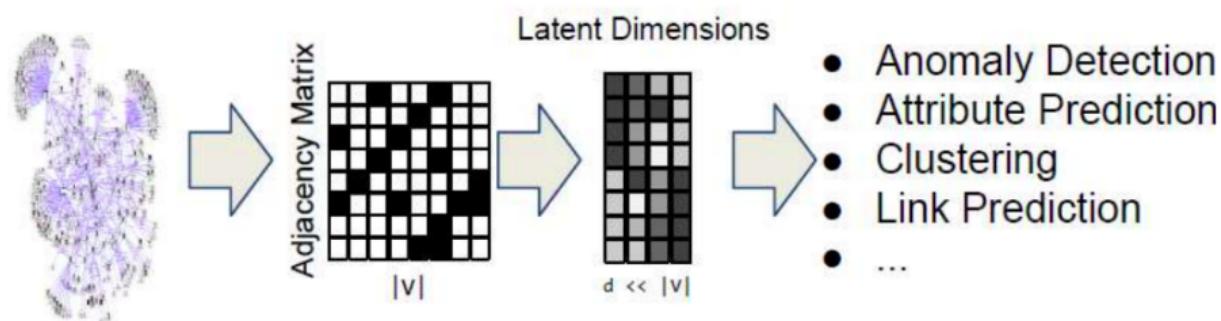
The slides of this part were derived from Jure Leskovec's slides - Stanford University

Feature Learning in Graphs



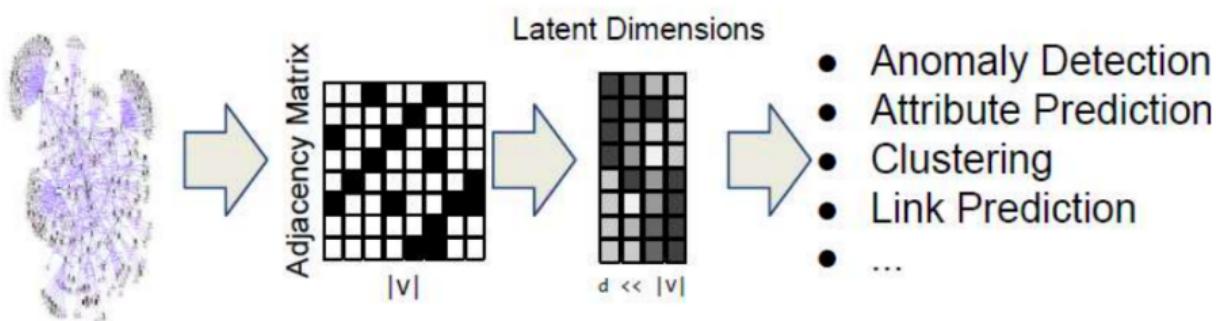
Why Learn Embedding?

- ▶ The goal is to map each node into a low-dimensional space.



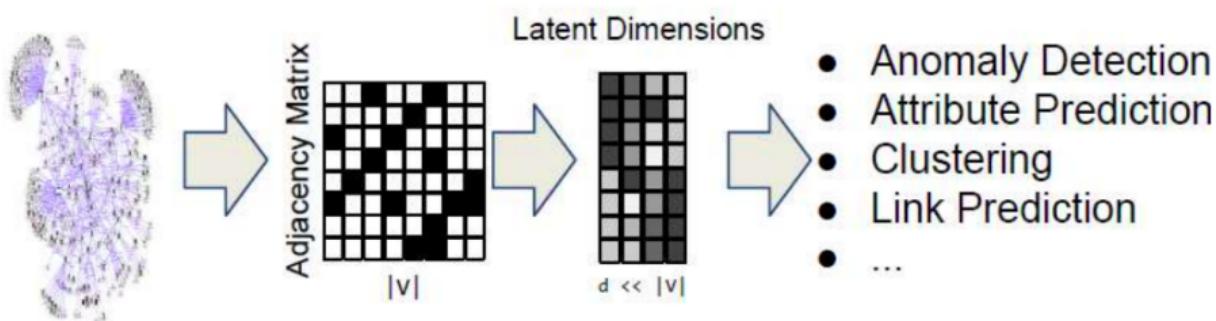
Why Learn Embedding?

- ▶ The goal is to map **each node** into a **low-dimensional space**.
 - **Representation** for nodes.



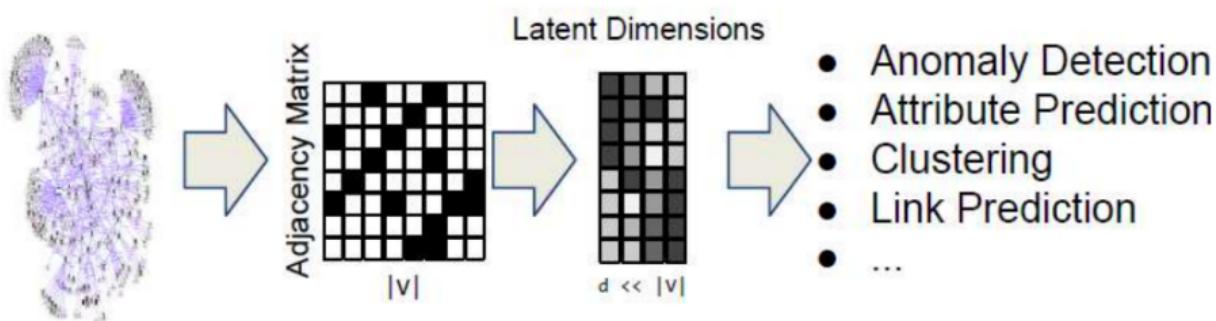
Why Learn Embedding?

- ▶ The goal is to map **each node** into a **low-dimensional space**.
 - **Representation** for nodes.
 - **Similarity** between nodes indicates **link strength**.

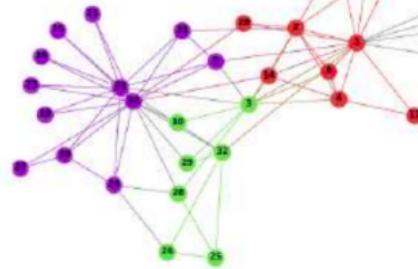


Why Learn Embedding?

- ▶ The goal is to map **each node** into a **low-dimensional space**.
 - **Representation** for nodes.
 - **Similarity** between nodes indicates **link strength**.
 - Encodes **network information** and generate node representation.

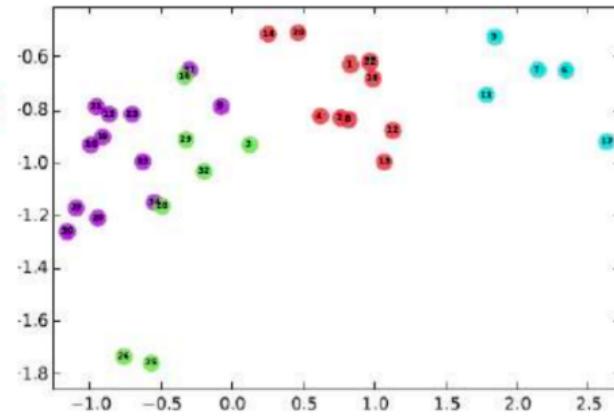


Example



Input

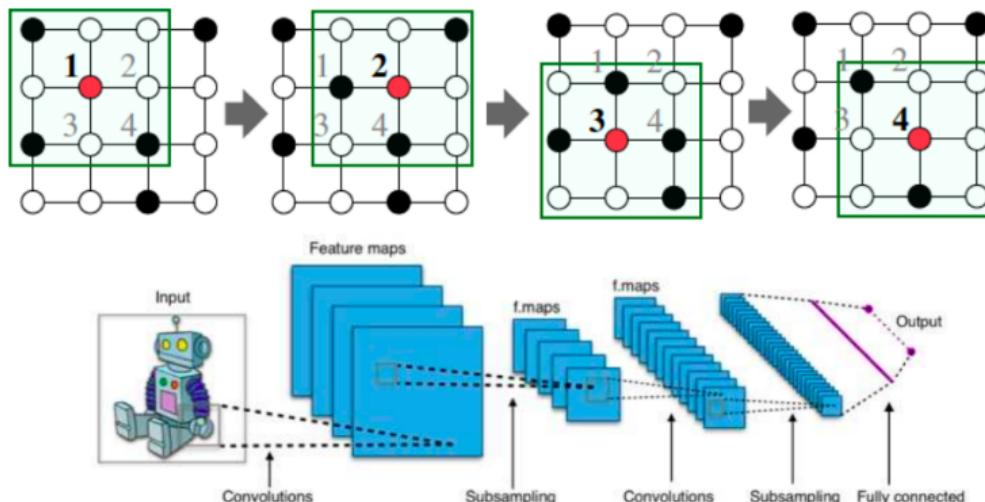
[Perozzi et al., DeepWalk: Online Learning of Social Representations, 2014]



Output

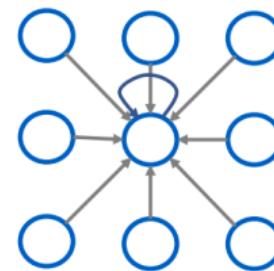
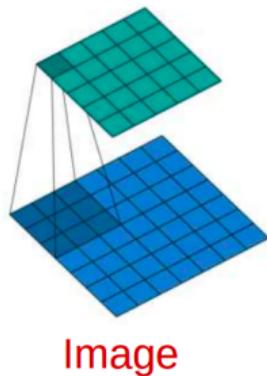
Idea: Convolutional Networks

- ▶ Goal is to generalize convolutions beyond simple lattices.
- ▶ Leverage node features/attributes (e.g., text, images).



From Images to Networks

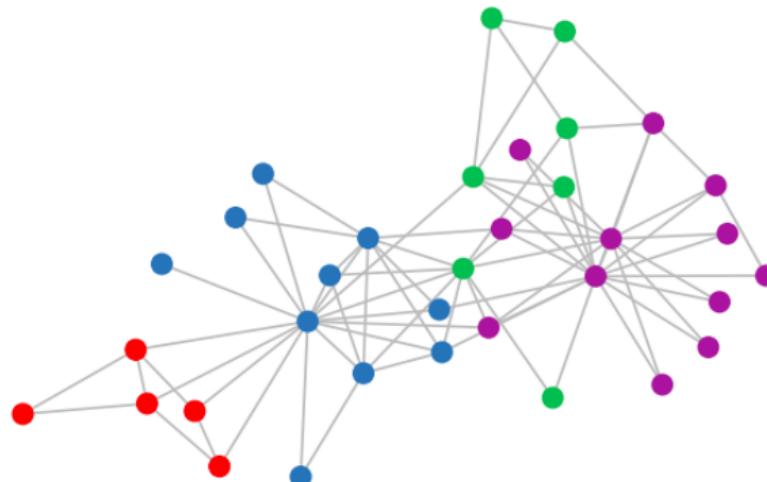
- ▶ Transform information at the **neighbors** and **combine it**:
 - Transform messages h_i from neighbors: $w_i h_i$
 - Add them up: $\sum_i w_i h_i$



Single CNN layer with 3×3 filter

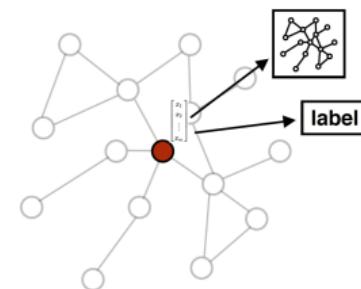
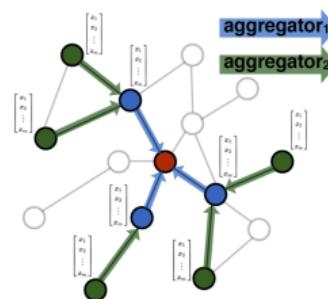
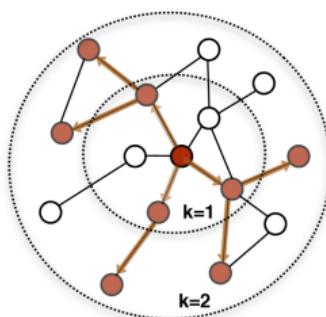
Real-World Graphs

- ▶ But what if your graphs look like this?



GraphSAGE (1/3)

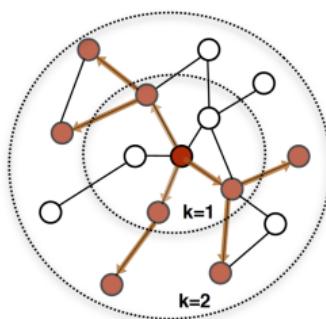
- ▶ GraphSAGE aggregates neighbouring node embeddings for a given node.



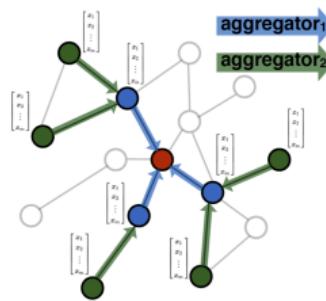
[<http://snap.stanford.edu/graphsage>]

GraphSAGE (1/3)

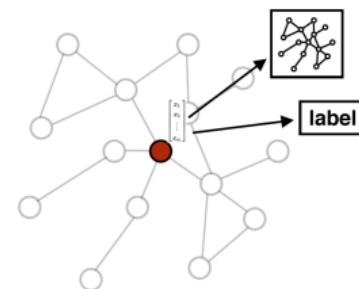
- ▶ GraphSAGE aggregates neighbouring node embeddings for a given node.
- ▶ The output of one round of GraphSAGE: new node representation for every node in the graph.



1. Sample neighborhood



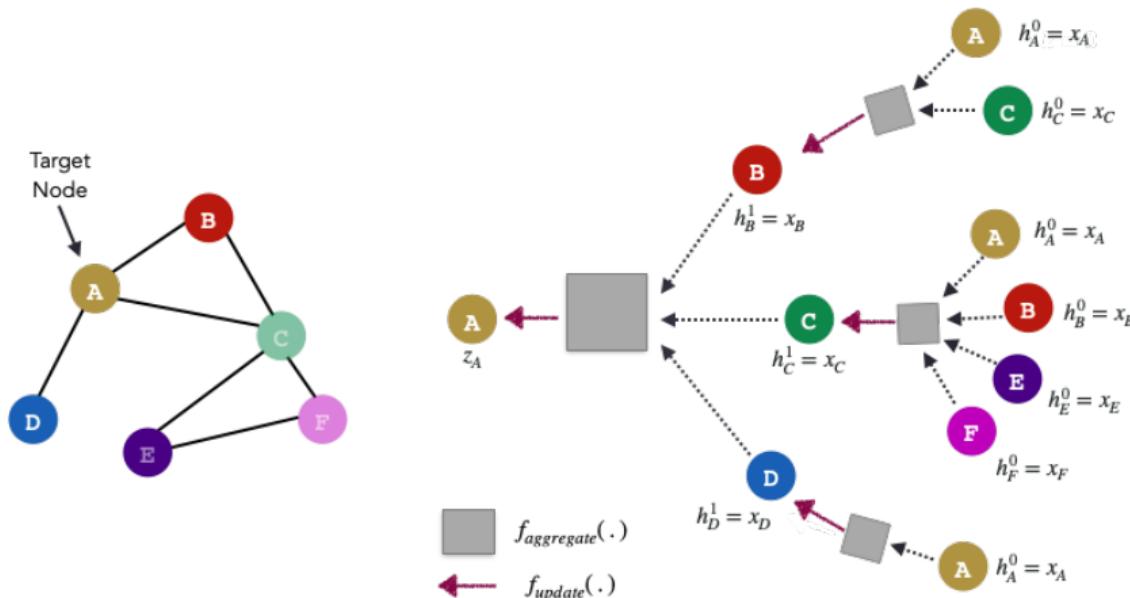
2. Aggregate feature information
from neighbors



3. Predict graph context and label
using aggregated information

[<http://snap.stanford.edu/graphsage>]

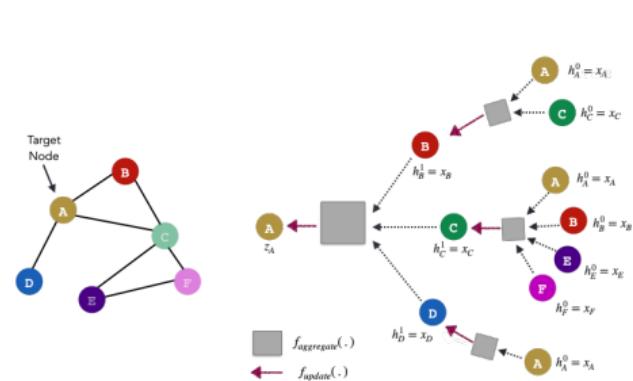
GraphSAGE (2/3)



[<https://mc.ai/ohmygraphs-graphsage-and-inductive-representation-learning-2>]

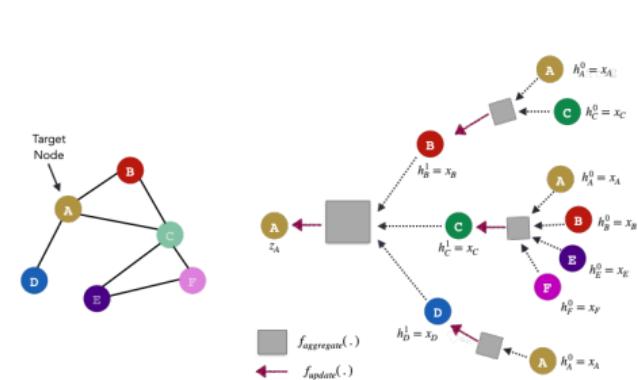
GraphSAGE (3/3)

► $h_{\mathcal{N}(v)}^1 = \max(f_a^i(h_u^1), \forall u \in \mathcal{N}(v))$



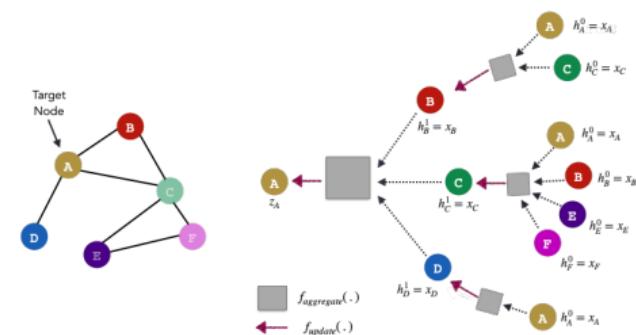
GraphSAGE (3/3)

- ▶ $h_{\mathcal{N}(v)}^1 = \max(f_a^i(h_u^1), \forall u \in \mathcal{N}(v))$
- ▶ $h_v^{l+1} = f_b^{l+1}(\text{concat}(h_v^l, h_{\mathcal{N}(v)}^l))$



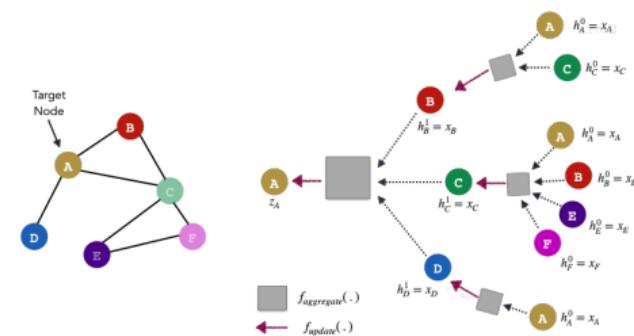
GraphSAGE (3/3)

- ▶ $h_{\mathcal{N}(v)}^1 = \max(f_a^i(h_u^1), \forall u \in \mathcal{N}(v))$
- ▶ $h_v^{l+1} = f_b^{l+1}(\text{concat}(h_v^l, h_{\mathcal{N}(v)}^l))$
- ▶ h_v : the hidden feature of v



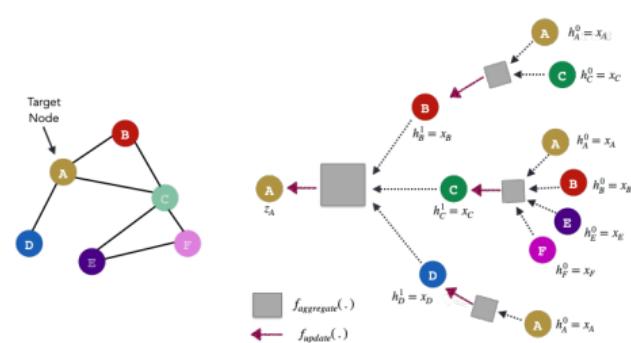
GraphSAGE (3/3)

- ▶ $h_{\mathcal{N}(v)}^1 = \max(f_a^i(h_u^1), \forall u \in \mathcal{N}(v))$
- ▶ $h_v^{l+1} = f_b^{l+1}(\text{concat}(h_v^l, h_{\mathcal{N}(v)}^l))$
- ▶ h_v : the hidden feature of v
- ▶ f_a and f_b : dense layers



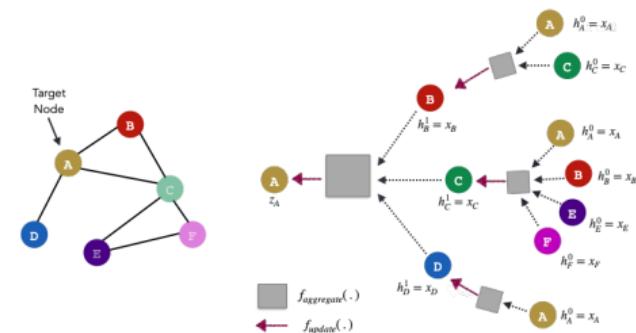
GraphSAGE (3/3)

- ▶ $h_{\mathcal{N}(v)}^1 = \max(f_a^i(h_u^1), \forall u \in \mathcal{N}(v))$
- ▶ $h_v^{l+1} = f_b^{l+1}(\text{concat}(h_v^l, h_{\mathcal{N}(v)}^l))$
- ▶ h_v : the hidden feature of v
- ▶ f_a and f_b : dense layers
- ▶ $\mathcal{N}(v)$: the neighbors of v



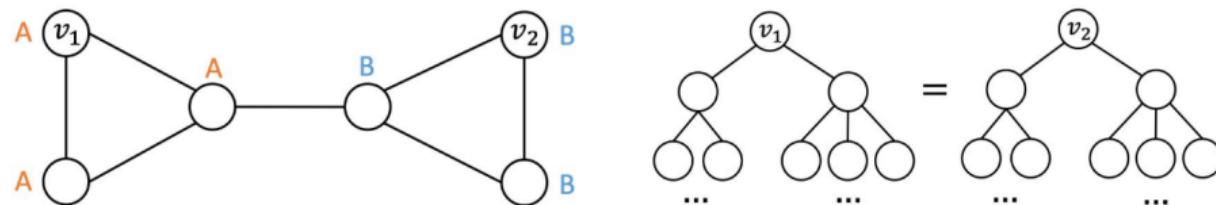
GraphSAGE (3/3)

- ▶ $h_{\mathcal{N}(v)}^1 = \max(f_a^i(h_u^1), \forall u \in \mathcal{N}(v))$
- ▶ $h_v^{l+1} = f_b^{l+1}(\text{concat}(h_v^l, h_{\mathcal{N}(v)}^l))$
- ▶ h_v : the hidden feature of v
- ▶ f_a and f_b : dense layers
- ▶ $\mathcal{N}(v)$: the neighbors of v
- ▶ $h_{\mathcal{N}(v)}$: the aggregated feature from the neighbors of v



GraphSAGE Shortcoming

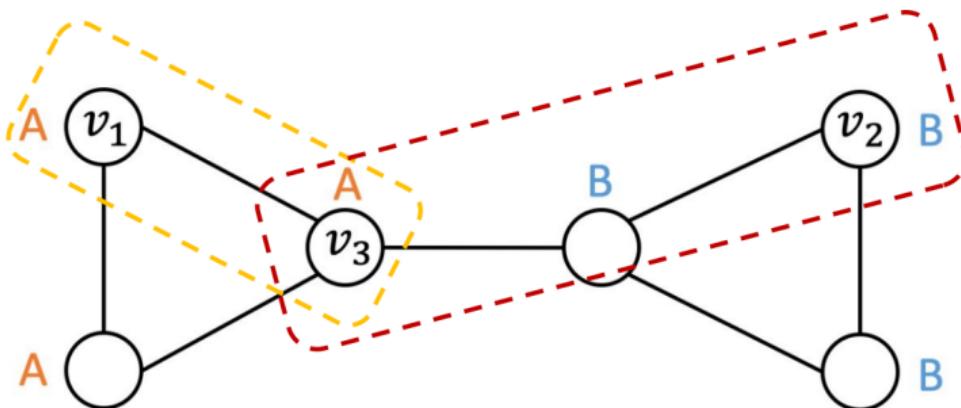
- ▶ Nodes with the **same neighborhoods** have the **similar embeddings**, regardless of their **location** in the graph?



[You et al., Position-aware Graph Neural Networks, 2019]

Position-aware Graph Neural Networks

- ▶ By adding anchor sets - we bypass that problem.



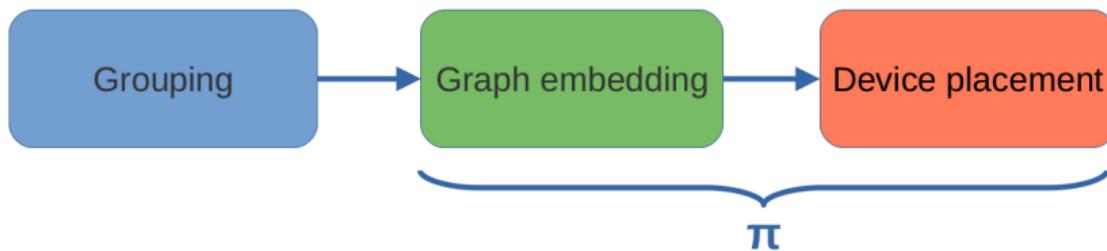
[Figure by Milko Mitropolitsky]



Solution 2

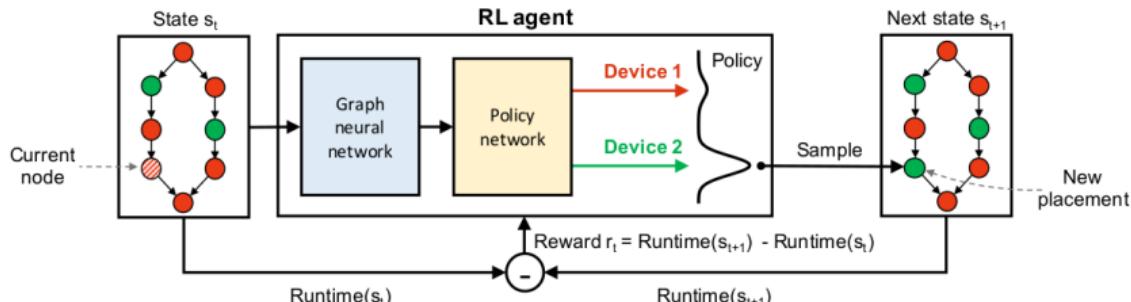
Addanki, et al., Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning, 2019

Device Placement Policy



Placeto System Overview

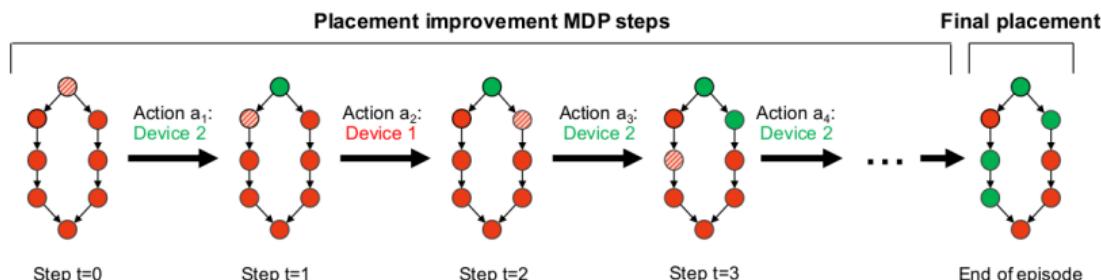
- ▶ Graph embedding
- ▶ Placement policy network



[Addanki, et al., Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning, 2019]

MDP Formulation (1/2)

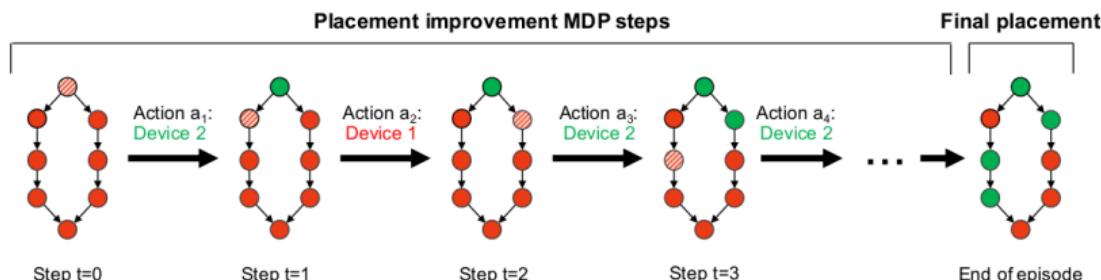
- ▶ Model the device placement as **Markov Decision Process (MDP)**.
- ▶ Initial state s_0 , consists of \mathcal{G} with an arbitrary device placement for each node group.



[Addanki, et al., Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning, 2019]

MDP Formulation (1/2)

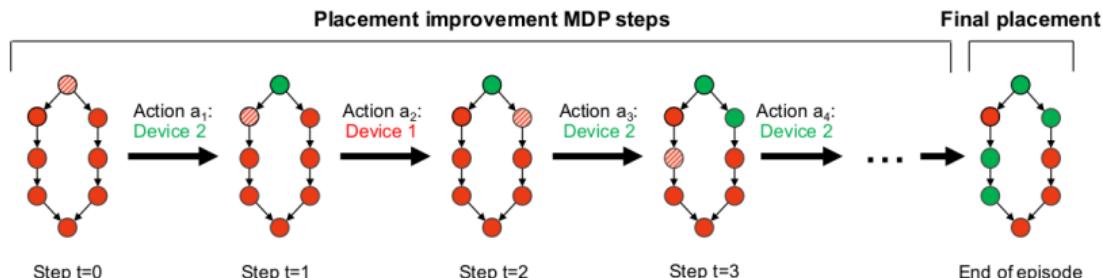
- ▶ Model the device placement as **Markov Decision Process (MDP)**.
- ▶ Initial state s_0 , consists of \mathcal{G} with an arbitrary device placement for each node group.
- ▶ Action in step t outputs a **new placement** for the t th node in \mathcal{G} based on s_{t-1} .



[Addanki, et al., Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning, 2019]

MDP Formulation (1/2)

- ▶ Model the device placement as **Markov Decision Process (MDP)**.
- ▶ Initial state s_0 , consists of \mathcal{G} with an arbitrary device placement for each node group.
- ▶ Action in step t outputs a **new placement** for the t th node in \mathcal{G} based on s_{t-1} .
- ▶ Episode ends in $|V|$ steps (V : set of nodes in \mathcal{G}).



[Addanki, et al., Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning, 2019]



MDP Formulation (2/2)

- ▶ Two approaches for assigning rewards.



MDP Formulation (2/2)

- ▶ Two approaches for assigning rewards.
- ▶ Approach 1: assign 0 reward at each intermediate RL step and the negative run time of the final replacement as final reward.

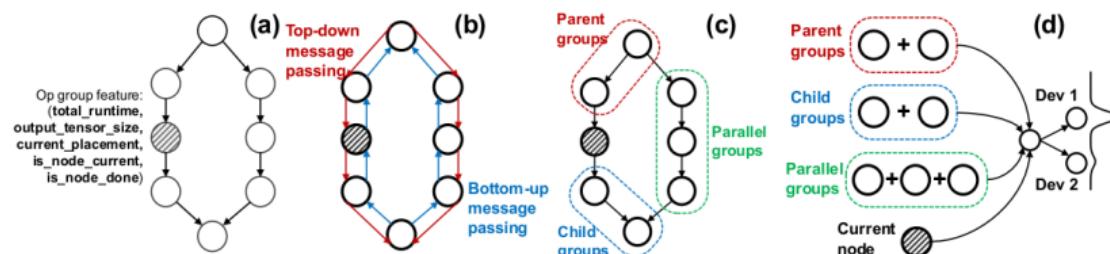


MDP Formulation (2/2)

- ▶ Two approaches for assigning rewards.
- ▶ Approach 1: assign 0 reward at each intermediate RL step and the negative run time of the final replacement as final reward.
- ▶ Approach 2: assign intermediate rewards $r_t = R(\mathcal{P}_{s_{t+1}}) - R(\mathcal{P}_{s_t})$

Graph Embedding

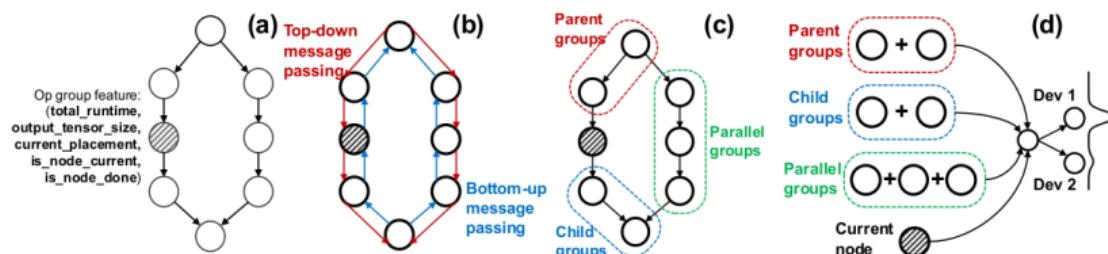
- ▶ Computing per-group attributes (a)



[Addanki, et al., Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning, 2019]

Graph Embedding

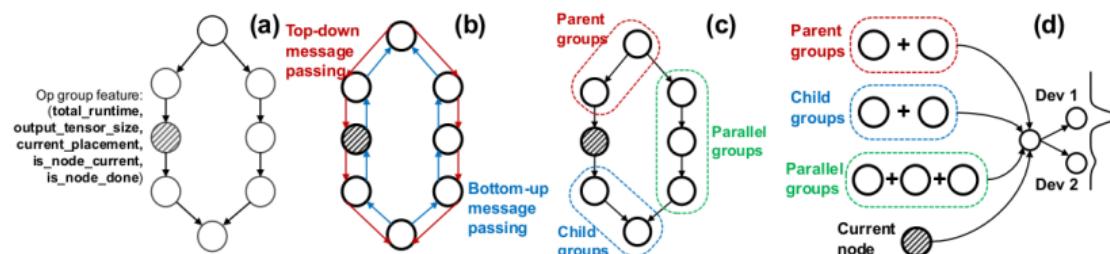
- ▶ Computing per-group attributes (a)
- ▶ Local neighborhood summarization (b)



[Addanki, et al., Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning, 2019]

Graph Embedding

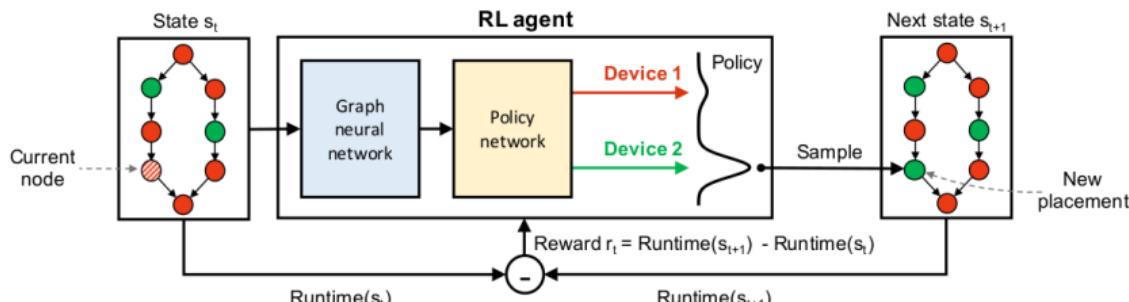
- ▶ Computing per-group attributes (a)
- ▶ Local neighborhood summarization (b)
- ▶ Pooling summaries (c)



[Addanki, et al., Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning, 2019]

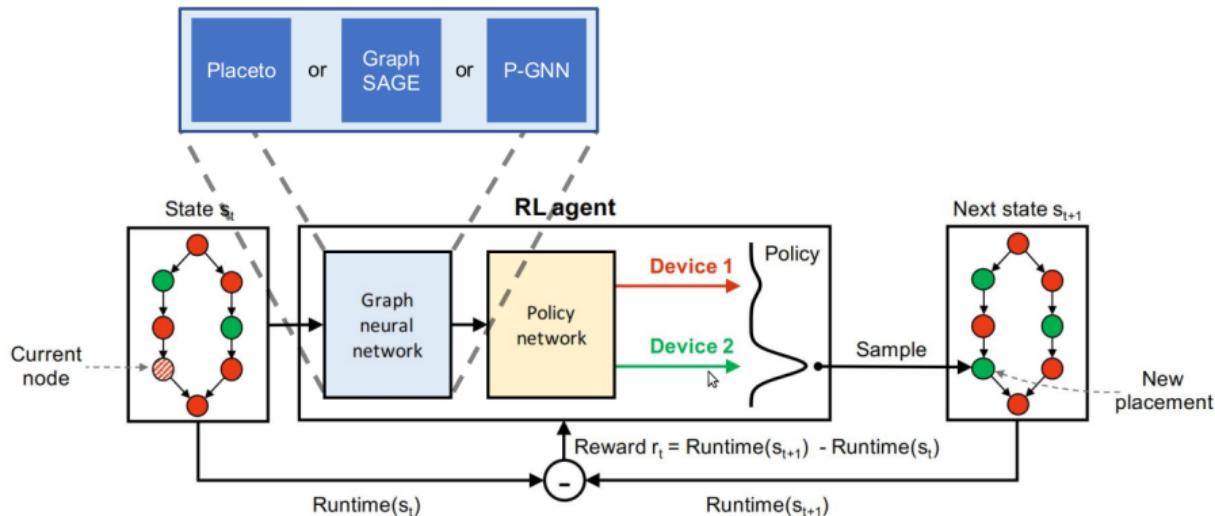
Placement Policy Network

- ▶ Implements the **MDP policy** using a three-layer fully connected neural network.
- ▶ Trains it using the **REINFORCE** policy-gradient algorithm.



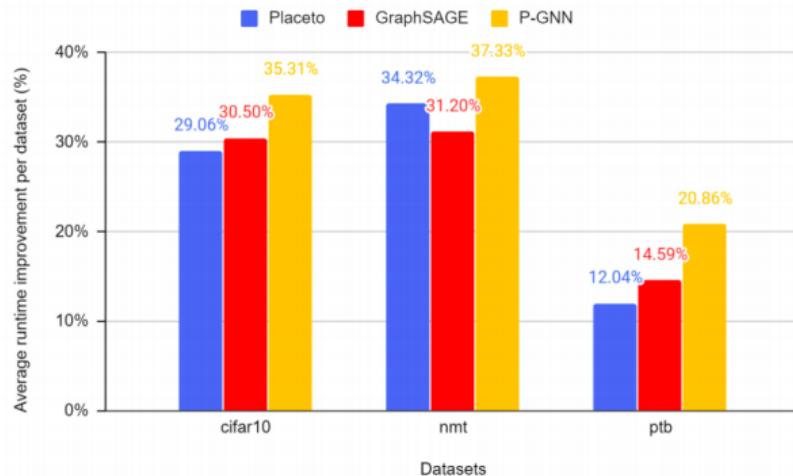
[Addanki, et al., Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning, 2019]

Graph Representation Matters in Device Placement (1/2)



[Mitropolitsky et al., Graph Representation Matters in Device Placement, 2020]

Graph Representation Matters in Device Placement (2/2)



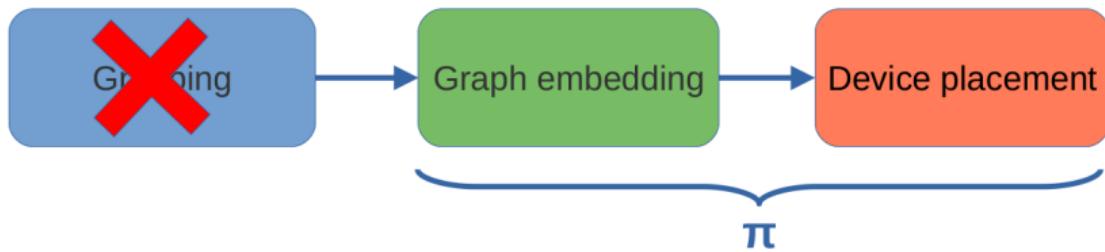
[Mitropolitsky et al., Graph Representation Matters in Device Placement, 2020]



Solution 3

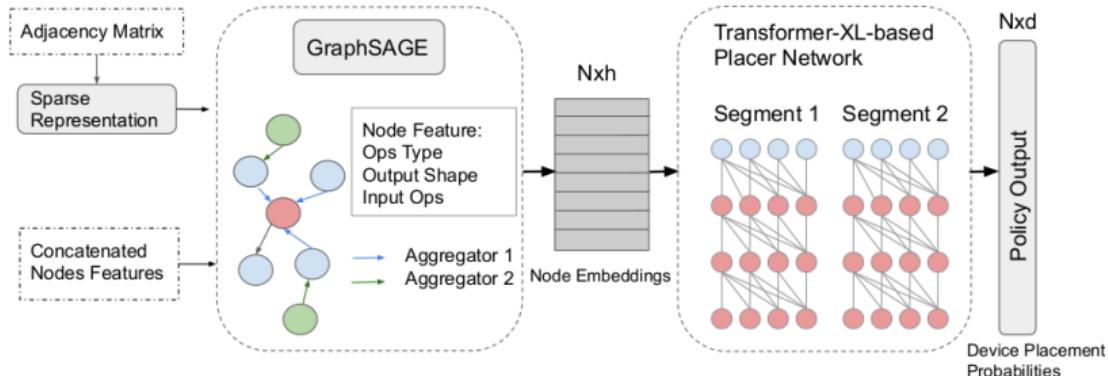
Zhou et al., A Single-Shot Generalized Device Placement for Large Dataflow Graphs, 2020

Device Placement Policy



GDP System Overview

- ▶ Uses a deep RL approach with **graph embeddings** and a **Transformer**.

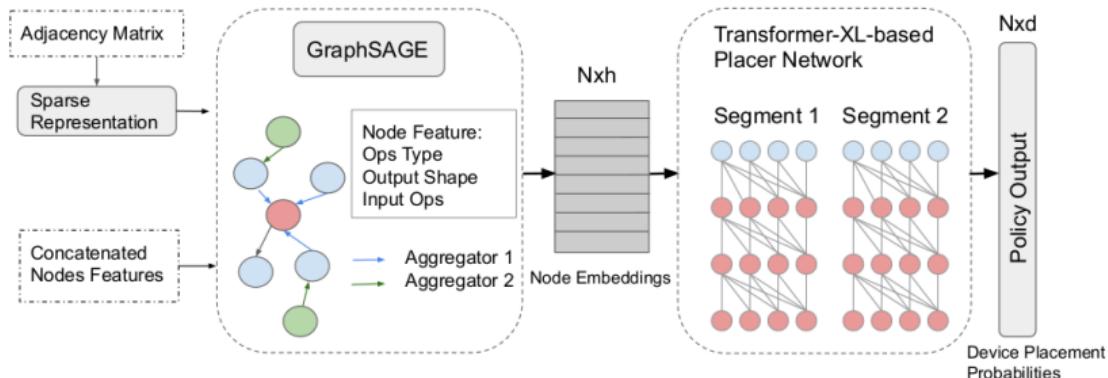


N: number of nodes, h: hidden Size, d: number of devices

[Zhou et al., GDP: Generalized Device Placement for Dataflow Graphs, 2019]

GDP System Overview

- ▶ Uses a deep RL approach with **graph embeddings** and a **Transformer**.
- ▶ Generalize to **unseen graphs**.

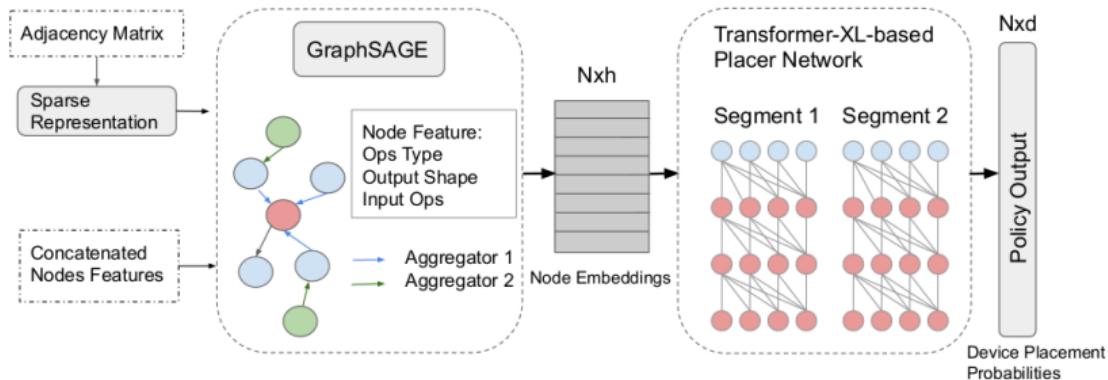


N: number of nodes, h: hidden Size, d: number of devices

[Zhou et al., GDP: Generalized Device Placement for Dataflow Graphs, 2019]

GDP System Overview

- ▶ Uses a deep RL approach with **graph embeddings** and a **Transformer**.
- ▶ Generalize to **unseen graphs**.
- ▶ Generates placement for the **whole graph in one step**, reducing training time.



N : number of nodes, h : hidden Size, d : number of devices

[Zhou et al., GDP: Generalized Device Placement for Dataflow Graphs, 2019]



Placement Policy Network (1/2)

- ▶ Conventional seq-to-seq models usually target short sequences, which requires grouping beforehand.



Placement Policy Network (1/2)

- ▶ Conventional seq-to-seq models usually target short sequences, which requires grouping beforehand.
- ▶ LSTM used in previous works is slower and more difficult to train than attention-based models.



Placement Policy Network (1/2)

- ▶ Conventional seq-to-seq models usually target short sequences, which requires grouping beforehand.
- ▶ LSTM used in previous works is slower and more difficult to train than attention-based models.
- ▶ GDP adopts segment-level recurrence introduced in Transformer-XL to capture long-term dependencies.



Placement Policy Network (1/2)

- ▶ Conventional seq-to-seq models usually target short sequences, which requires grouping beforehand.
- ▶ LSTM used in previous works is slower and more difficult to train than attention-based models.
- ▶ GDP adopts segment-level recurrence introduced in Transformer-XL to capture long-term dependencies.
- ▶ The key is to cache (with gradient flows disabled) and reuse the hidden states of previous segments.

Placement Policy Network (2/2)

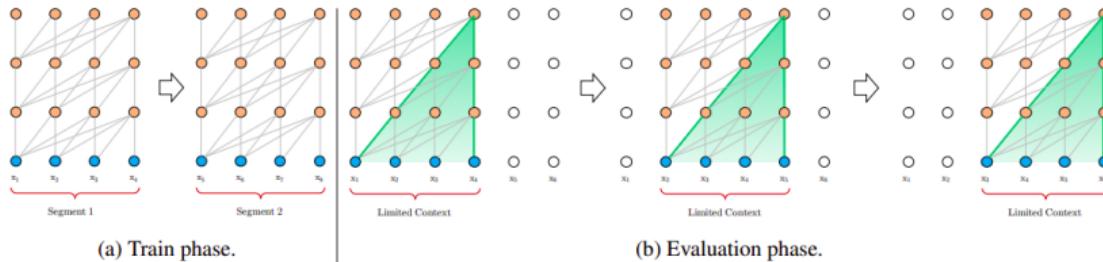


Figure 1: Illustration of the vanilla model with a segment length 4.

Placement Policy Network (2/2)

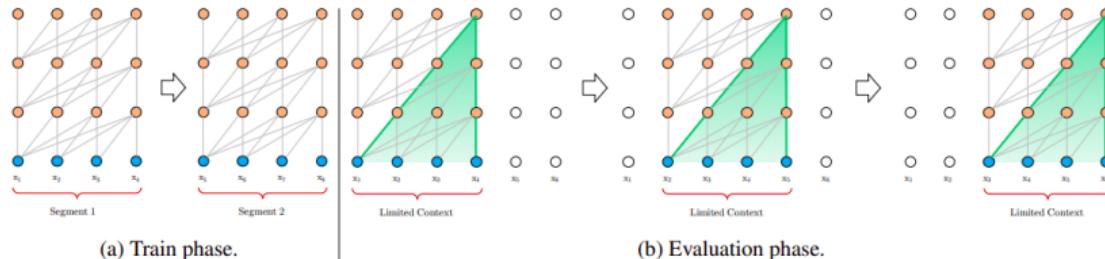


Figure 1: Illustration of the vanilla model with a segment length 4.

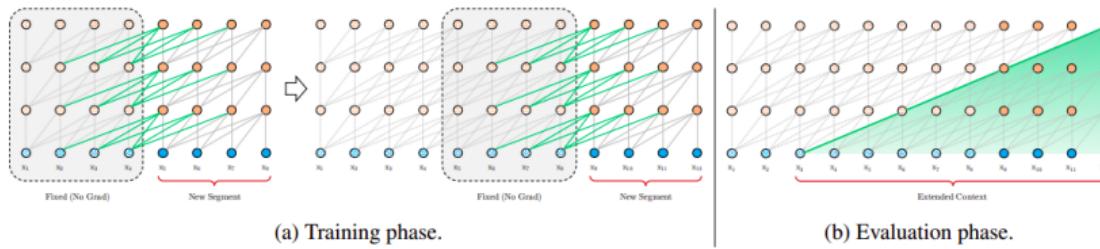


Figure 2: Illustration of the Transformer-XL model with a segment length 4.

[Z. Dai et al., Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context, 2019]



Batch Training

- ▶ Prior works focus on a single graph only.



Batch Training

- ▶ Prior works focus on a single graph only.
- ▶ In GDP, the RL objective is defined to simultaneously reduce the expected runtime of the placements over a set of N dataflow graphs.



Batch Training

- ▶ Prior works focus on a single graph only.
- ▶ In GDP, the RL objective is defined to simultaneously reduce the expected runtime of the placements over a set of N dataflow graphs.
- ▶ $J(w) = \mathbb{E}_{G \sim \mathcal{G}, \mathcal{P} \sim \pi(\mathcal{P}|G, w)} [R(\mathcal{P})|G] = \frac{1}{N} \sum_G \mathbb{E}_{\mathcal{P} \sim \pi(\mathcal{P}|G, w)} [R(\mathcal{P})|G]$



Summary



Summary

- ▶ Model parallelization and device placement
- ▶ Hierarchical device placement
- ▶ Placeto
- ▶ GDP



Reference

- ▶ Mayer, R. et al., The TensorFlow Partitioning and Scheduling Problem, 2017
- ▶ Mirhoseini et al., Device Placement Optimization with Reinforcement Learning, 2017
- ▶ Mirhoseini et al., A Hierarchical Model for Device Placement, 2018
- ▶ Addanki, et al., Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning, 2019
- ▶ Zhou et al., GDP: Generalized Device Placement for Dataflow Graphs, 2019



Questions?