Runtime Performance prediction of deep learning models with GNN

Yanjie Gao, Xianyu Gu, Hongyu Zhang, Haoxiang Lin, Mao Yang
2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)

Presenter

Dipak Acharya
University of North Texas
Denton TX

Overview

- Deep learning (DL) models are highly configurable.
- Require multiple training jobs to find the optimal settings.

- Runtime performance, such as GPU memory usage and training time, are very important attributes useful for improving the model quality and boosting development productivity.
    - E.g. Knowledge of these attributes may allow us to avoid *out-of-memory* (OOM) exceptions in DL training.

Challenges

- Complex nature of DL models and programming paradigms
- Many hidden features that are difficult to extract
- Large configuration space.

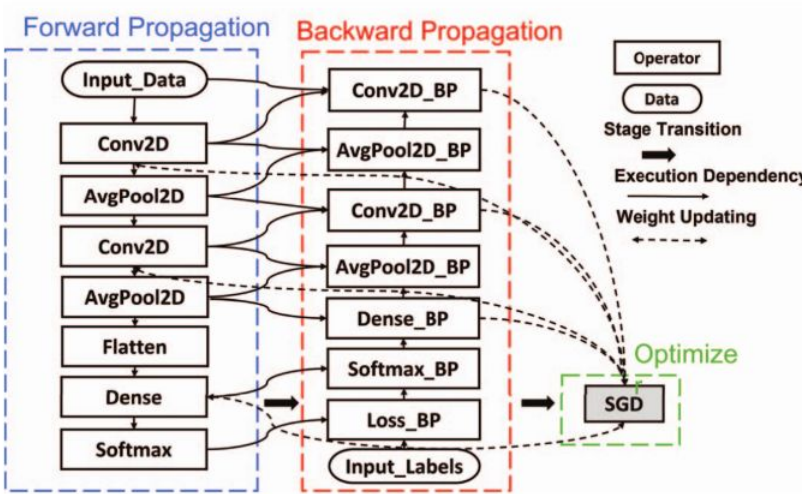
DNNPerf

- ***DNNPerf*** is a machine learning tool using a Graph Neural Network (GNN) for predicting DL model runtime performance.
- It represents DL models as computational graphs with detailed node and edge features.
- It Incorporates a new *Attention Based Node-Edge Encoder* based on Graph Attention Network (GAT) and Edge Enhanced Graph Neural Network (EGNN) concepts.
- ***DNNPerf*** is trained on real and synthetic DL model configurations.
- Achieves mean relative errors of 7.4% and 13.7% in predicting training time and GPU memory consumption, outperforming other methods.

Background

DL Models and computational graphs

- DL models are formalized by frameworks as tensor-oriented computational graphs.
- Each node represents a mathematical operation and edges indicate execution dependencies between nodes.
- For training the framework constructs computation graph and performs iterative forward and backward propagation to calculate gradient of parameters.
- An optimizer will use it to update the parameters(weights) for minimum loss.



Background

Because of variation in the number of neighbours in each node, common neural architectures such as Convolutional Neural Networks(CNN) and Recurrent Neural Network(RNN) cannot be used with graph data.

GNNs

- Convert each node to fixed sized vectors (embeddings) using graph layers
- Node embeddings aggregate information of neighbors within certain hops
- By stacking *n* layers, we can aggregate information from *n-hop* neighbors
- This *n* value is a GNN specific hyperparameter
- To accommodate different node sizes, a fixed size vector (embedding) is created for graph
- Use embeddings vector on traditional machine learning algorithm such as MLP

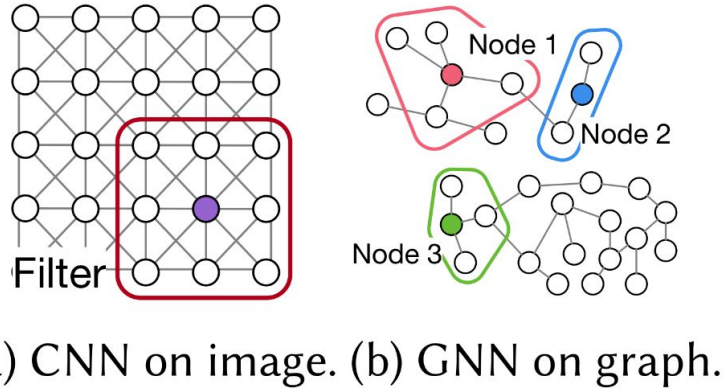


Fig. Comparison of CNN and GNN

Source: Yang et al. 2022. Prediction of the Resource Consumption of Distributed Deep Learning Systems. *Proc. ACM Meas. Anal. Comput. Syst. 6, 2,* Article 29 (June 2022), 25 pages. https://doi.org/10.1145/3530895

Problem Formulation

A DL model can be represented as a Directed Acyclic Graph (DAG)

$$\mathcal{M} = \langle \{u_i\}_{i=1}^{n}, \{e_{ij} = (u_i, u_j)\}_{i,j \in [1,n]}, \{hp_k\}_{k=1}^{m} \rangle .$$

Here u_i is a node,
e_{ij} is a edge from u_i to u_j
h_{pk} is a hyperparameter of some operator

A single model configuration is denoted as

$$\mathcal{M}(b_1 \in B_1, b_2 \in B_2, \cdots, b_m \in B_m)$$

Model Configuration Space is denoted as

$$\Delta_{\mathcal{M}} = \{ \mathcal{M}(b_1, b_2, \cdots, b_m) \mid b_k \in B_k \text{ for } k \in [1, m] \} .$$

Problem Formulation…

Runtime configuration describes the environment such as the device type and FLOPs
Runtime configuration space is denoted by S

Performance function is defined as a regression task based on the model configuration and the runtime configuration

$$f_i : \Delta_{\mathcal{M}} \times \mathcal{S} \to \mathbb{R}\,.$$

Workflow

Input: DL model file, model configuration specification and runtime specification

Output: runtime performance prediction (usually a real value)

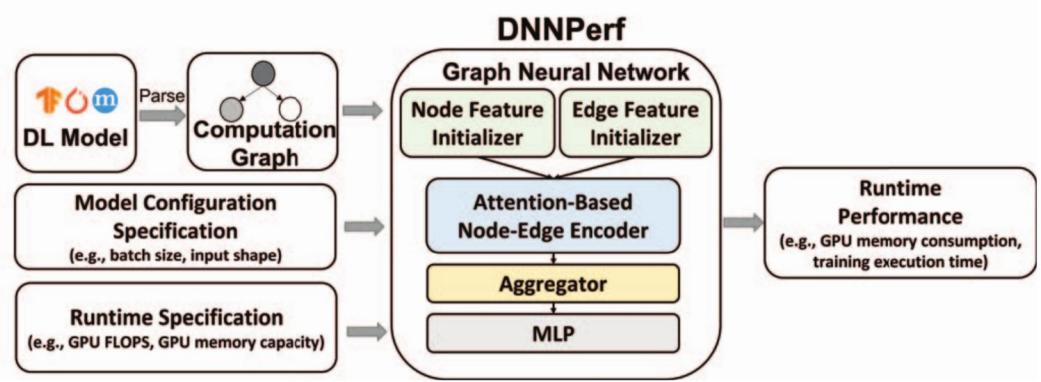


Fig. 3: Workflow of DNNPerf.

Model Encoding: Nodes encoding

- **Initial Node Encoding**

$$h = F(h^t \parallel h^p \parallel h^d \parallel h^c \parallel h^r)$$

$\parallel$ is the vector concatenation operation, and F is the function for raw feature encoding and vector shape alignment

TABLE I
NODE FEATURES.

Category	Name	Description
Operator (h^t)	Operator Type	Type of the operator (e.g., `Conv2D` and `AvgPool2D`)
Hyperparameter (h^p)	Hyperparameter	Type and value of each hyperparameter of the operator (e.g., kernel size and channel size)
Tensor (h^d)	Weight Tensor Size	Sizes of weights (aka learnable parameters) and weight gradients (computed under backward propagation for updating weights)
	In/Out Tensor Size	Sizes of inputs, outputs, and output gradients (computed under backward propagation for calculating weight gradients)
	Temporary Tensor Size	Sizes of temporary variables used by the operator
Computation Cost (h^c)	FLOPs	Number of floating-point operations of the operator
Device (h^r)	FLOPS	Peak floating-point operations per second
	Memory Capacity	Total amount of the device memory

Model Encoding: Edge encoding

- **Initial Edge encoding**

$$e = e^t \parallel e^d \parallel e^r$$

TABLE II
EDGE FEATURES.

Category	Name	Description
Edge (e^t)	Edge Type	Type of the edge (“forward” or “backward”)
Tensor (e^d)	Tensor Size	Size of the delivered tensor
Device (e^r)	Bandwidth	Bandwidth for accessing the delivered tensor

Model Encoding: Normalization

To Normalize the Node and Edge Features *MinMaxScaler* is used

Node feature Normalization

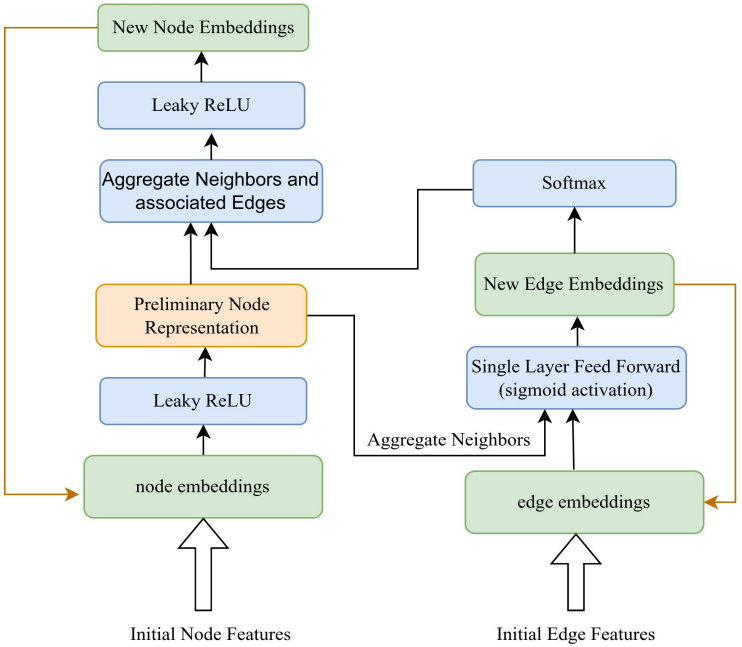
$$S_i = \{h_{g,u,i} \mid g \in DS \wedge u \in g\},$$
$$\hat{h}_{g,u,i} = \frac{h_{g,u,i} - \min S_i}{\max S_i - \min S_i}.$$

Edge feature Normalization

$$S_i = \{e_{g,l,i} \mid g \in DS \wedge l \in g\},$$
$$\hat{e}_{g,l,i} = \frac{e_{g,l,i} - \min S_i}{\max S_i - \min S_i}.$$

Attention Based Node Edge Encoder (ANEE)

- ANEE is a encoder based on Graph Attention Network (GAT)
- It Utilizes both node and edge features to generate a rich set of embeddings
- It starts from the initial normalized features and on each round produce a new set of embeddings



Attention Based Node-Edge Encoder (ANEE)

Attention Based Node Edge Encoder (ANEE)

First, ANEE computes a preliminary result of $h^i_{g,u}$ $\overline{h}^i_{g,u}$

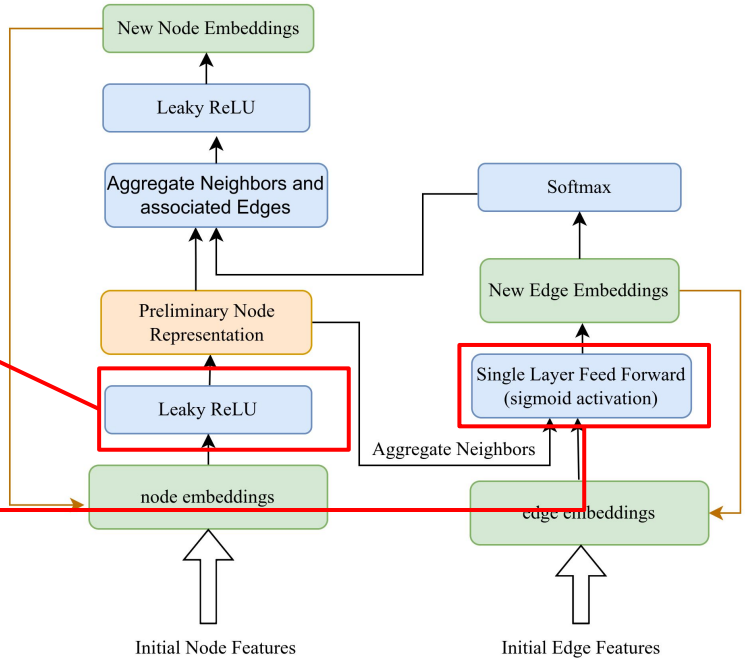
$$\texttt{LeakyReLU}(x) = \max(0, x) - \alpha \times \max(0, -x)\,,$$
$$\overline{h}^i_{g,u} = \texttt{LeakyReLU}(\mathbf{W}_u \times h^{i-1}_{g,u})\,.$$

α is the slope coefficient (default 0.01)

Then, the Update function for $e^i_{g,l}$ is given as follows

$$e^i_{g,l} = \sigma(a^T \times (\overline{h}^i_{g,s} \parallel \overline{h}^i_{g,d}) \times \mathbf{W}_e \times e^{i-1}_{g,l})\,.$$

Here a is the weight matrix of single layer feedforward neural network



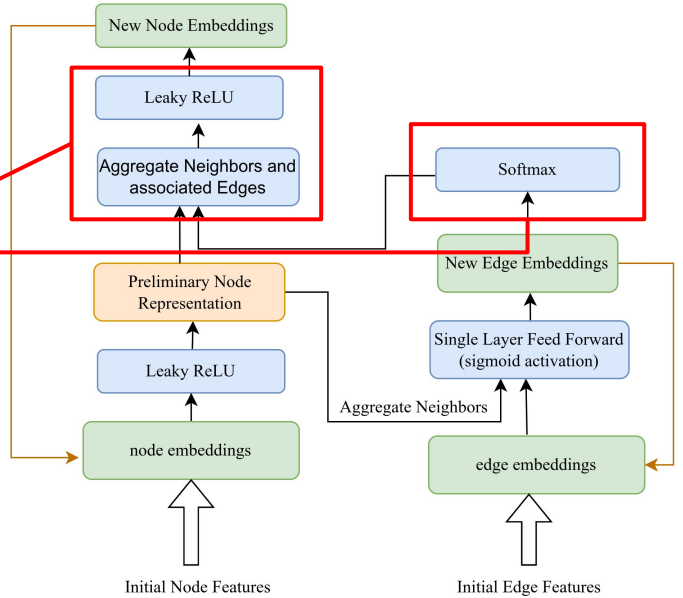
Attention Based Node-Edge Encoder (ANEE)

Attention Based Node Edge Encoder (ANEE)

Then aggregate the neighboring node and edges to calculate $h^i_{g,u}$ as follows

$$f(u', l') = \texttt{Softmax}(\mathbf{W}_m \times e^i_{g,l'}) \times \overline{h}^i_{g,u'},$$
$$h^i_{g,u} = \texttt{LeakyReLU}(\sum_{l'=(u',u)} f(u', l')).$$

Then finally, *DNNPerf* performs a global aggregation by summing up all the node feature vectors. The result is fed to an MLP predictor layer to generate the runtime performance value.



Attention Based Node-Edge Encoder (ANEE)

Experimental setup: Datasets

- **HPO Datasets**
  - Representative deep learning models from the TensorFlow-Slim model library
  - Employs a random strategy to generate model configurations.
  - Tuned hyperparameters include batch size, number of input channels, input height, and input width with specific intervals and domains defined for each model

- **NAS Datasets**
  - Generated with NAS search with model skeleton composed of various operator cells and hyperparameters.
  - Operators include Conv2D, Dense, MaxPool2D etc.
  - Key hyperparameters include batch size, input dimensions, number of input/output channels, kernel sizes for Conv2D and MaxPool2D, and the number of units for the Dense operator

Experimental setup:Baselines

For Evaluation of *DNNPerf*, it is compared with following baselines

- BiRNN (Bidirectional RNN)
- ARNN (Adjacency BiRNN)
- MLP (Multilayer Perceptron)
- GBDT (Gradient Boost Decision Tree)
- BRP-NAS (Binary Relation Predictor-based NAS)

The Evaluation section will compare the results of *DNNPerf* with all these approaches

Evaluation

Evaluation Metrics: Mean Relative Error(MRE**)** and Root-Mean Square error(RMSE**)**

RMSE values are in Gigabyte(GB) for memory consumption and millisecond(ms) for training time prediction

The overall results shows *DNNPerf* outperforming all the baselines

Evaluation was done based on 3 important criterias

RQ1: How effective is *DNNPerf* in predicting runtime performance?

RQ2: How general is *DNNPerf* to unseen DL models?

RQ3: How effective is *DNNPerf* in the ablation study?

Evaluation:Effectiveness

DNNPerf has both better MRE and RMSE on majority of the baseline benchmarks

This can be attributed to *DNNPerf*'s ability to capture more richer and diverse features from the node and edges

The overall MRE/RMSE improvement over baselines

- 14.0%/23.4 to 60.5%/190 for training time prediction
- 4.0%/0.3 to 50.4%/3.1 for GPU memory consumption prediction.

OVERALL EXPERIMENTAL RESULTS.

Model Name	Prediction of Training Time						
	Metrics	DNNPerf	BiRNN	ARNN	MLP	GBDT	BRP-NAS
Overall	MRE (%)	**7.4**	20.8	20.9	18.0	31.0	95.2
	RMSE (ms)	**58.5**	151.4	113.9	106.2	122.6	335.4
HPO	MRE (%)	**7.8**	21.6	22.2	19.4	33.1	86.7
	RMSE (ms)	**51.9**	107.7	95.2	94.7	120.4	294.3
NAS	MRE (%)	**6.3**	18.4	16.9	13.5	24.4	122.3
	RMSE (ms)	**75.7**	242.6	159.5	136.4	129.3	441.0

Model Name	Prediction of GPU Memory Consumption						
	Metrics	DNNPerf	BiRNN	ARNN	MLP	GBDT	BRP-NAS
Overall	MRE (%)	**13.7**	22.3	25.5	21.5	15.6	33.7
	RMSE (GB)	**1.8**	2.8	2.9	2.3	2.2	4.1
HPO	MRE (%)	**13.2**	22.1	26.7	22.1	14.2	31.5
	RMSE (GB)	**1.8**	2.8	3.0	2.4	2.1	4.2
NAS	MRE (%)	**15.3**	22.8	21.7	19.4	20.2	40.5
	RMSE (GB)	**1.8**	2.8	2.4	2.0	2.4	4.0

Evaluation: Generalization

For this a test set consisting of the model configurations of AlexNet, VGG, OverFeat, ResNet-V2, and GRU were used (all unseen)

The improvement in MRE/RMSE ranges from 10.9%/40.4 to 83.5%/253.9 for the training time prediction task and from 0.2%/0.4 to 12.4%/2.2 for the GPU memory consumption prediction task

TABLE VII
EXPERIMENTAL RESULTS ON UNSEEN MODELS.

Model Name	Prediction of Training Time						
	Metrics	DNNPerf	BiRNN	ARNN	MLP	GBDT	BRP-NAS
Overall	MRE (%)	**7.7**	19.8	19.8	18.6	34.5	91.2
	RMSE (ms)	**55.1**	111.8	95.5	99.1	131.2	309.0
AlexNet	MRE (%)	**11.9**	22.5	18.4	21.6	85.4	291.1
	RMSE (ms)	**13.6**	27.1	28.6	26.5	90.4	264.6
VGG	MRE (%)	**7.3**	13.3	10.1	16.8	12.6	41.6
	RMSE (ms)	**84.2**	164.6	92.3	154.4	148.6	339.0
OverFeat	MRE (%)	**7.5**	20.7	24.9	16.2	58.5	65.4
	RMSE (ms)	**24.3**	90.7	77.9	77.6	178.6	154.4
ResNet-V2	MRE (%)	**5.7**	7.2	10.6	5.9	9.9	65.0
	RMSE (ms)	91.1	98.1	118.1	**65.9**	175.2	584.0
GRU	MRE (%)	**6.3**	28.2	28.7	26.0	15.2	29.3
	RMSE (ms)	**29.7**	118.7	118.9	105.8	59.1	130.7

Model Name	Prediction of GPU Memory Consumption						
	Metrics	DNNPerf	BiRNN	ARNN	MLP	GBDT	BRP-NAS
Overall	MRE (%)	**13.3**	21.6	25.7	23.1	13.5	23.5
	RMSE (GB)	**1.9**	2.8	3.0	2.6	2.3	4.1
AlexNet	MRE (%)	13.5	30.3	42.6	38.2	16.2	**11.2**
	RMSE (GB)	**1.3**	2.6	3.5	3.2	1.5	0.9
VGG	MRE (%)	**15.8**	17.9	17.9	16.8	16.5	29.0
	RMSE (GB)	**2.3**	3.2	2.6	2.4	3.0	4.6
OverFeat	MRE (%)	**13.6**	28.7	35.1	34.5	14.0	13.6
	RMSE (GB)	1.2	2.5	3.0	3.0	1.3	**1.1**
ResNet-V2	MRE (%)	**8.3**	14.3	15.1	9.6	8.4	30.2
	RMSE (GB)	**1.6**	2.8	2.9	1.7	1.8	4.7
GRU	MRE (%)	13.8	18.0	20.1	17.8	**12.1**	30.3
	RMSE (GB)	**2.3**	3.0	3.0	2.4	2.8	5.5

Evaluation: Ablation

Several ablation studies were done to validate the choices made for *Dnsperf*

DNNPerf mostly outperforms other approaches in terms of MRE and RMSE demonstrating its efficiency and stability

ABLATION STUDY.

Ablation Description	Training Time		GPU Memory Consumption	
	MRE (%)	RMSE (ms)	MRE (%)	RMSE (GB)
DNNPerf	**7.4**	58.5	**13.7**	**1.8**
DNNPerf-GAT	11.7	101.3	17.1	2.0
DNNPerf-StandardScaler	11.8	**56.0**	15.0	2.0
DNNPerf-NoTensorCost	15.9	91.7	24.4	3.3
DNNPerf-ConcatEdge	8.0	60.5	15.2	1.9
DNNPerf-AvgReadout	20.9	100.9	19.8	2.4

Contributions

- Introduces a GNN-based approach for DL model runtime performance prediction.
- Design rich set of node and edge features to capture performance related factors. Also propose a novel attention based node-edge encoder for the computational graph of a deep learning model.
- Implements and validates a tool called ***DNNPerf*** for runtime performance prediction of deep learning models, demonstrating its effectiveness.

Questions