Learning Compact DNN Models for Embedded Vision

Shuvra S. Bhattacharyya
University of Maryland, College Park, USA and
INSA/IETR Rennes, France

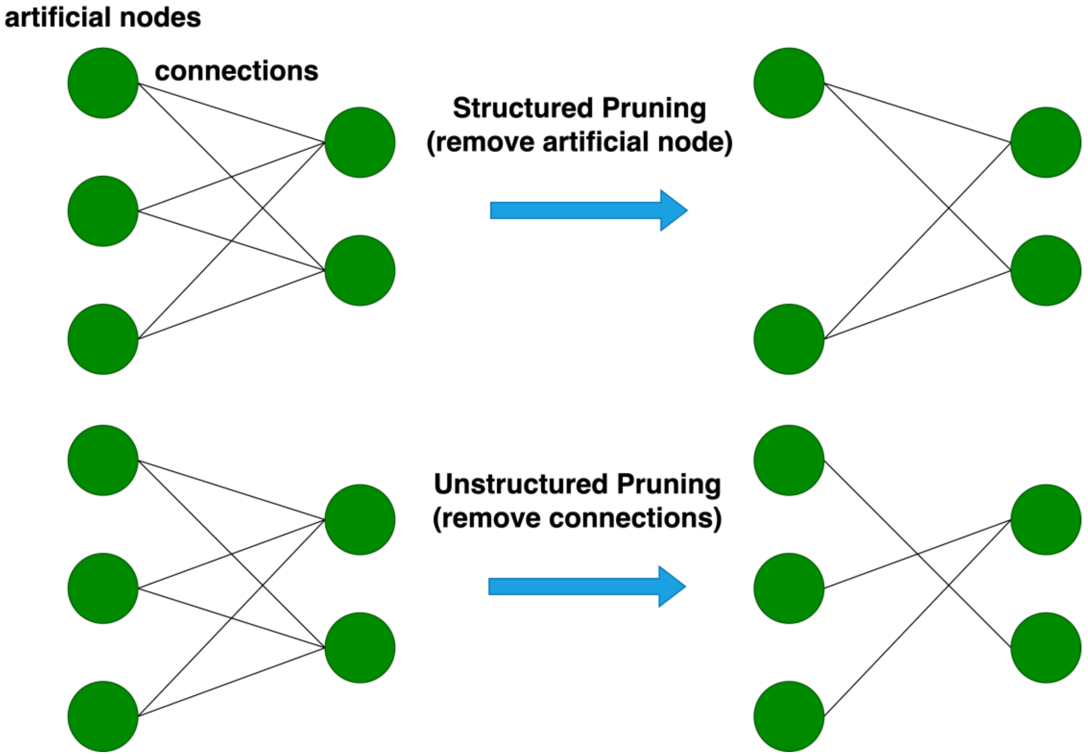
With contributions from Xiaomin Wu and Rong Chen

Popular Methods to Compress DNN Models

- Pruning:
  - Remove neurons or parameters that provide little or no contribution to inference accuracy
- Distillation:
  - Transfer knowledge from a large model to a small model
- Neural Architecture Search:
  - Optimize the number, types and connectivity of network layers

Pruning: Structured and Unstructured

Multilayer Perceptron: hidden layer example



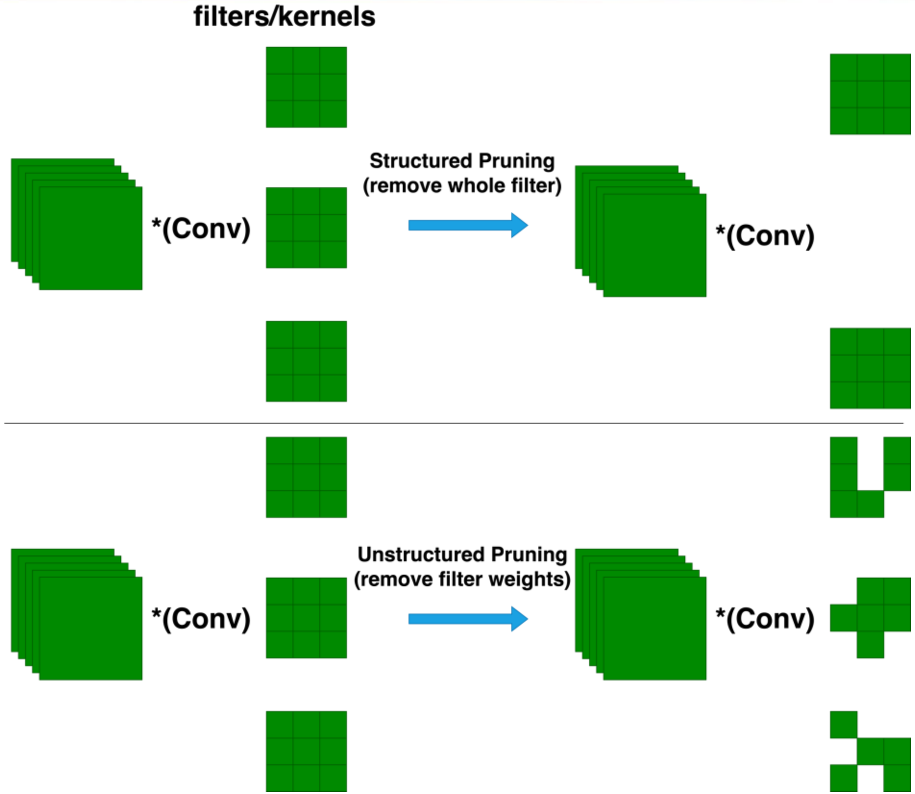
- Implementation-friendly
- Supports common ML libraries

- More general
- Needs specially-designed hardware/ software for sparse computation

UNIVERSITY OF MARYLAND

Pruning: Structured and Unstructured

CNN-layer filter example



- Implementation-friendly
- Supports common ML libraries

- More general
- Needs specially-designed hardware/ software for sparse computation

Previously-developed Pruning Methods

- Deep Compression [Han 2015]
  - Uses weight threshold to prune. Leads to unstructured network architecture.
- Inference-time channel reduction without retraining [He 2017]
  - Applies a criterion based on Lasso Regression.
- ThiNet — weight-magnitude-based structured pruning [Luo 2017]
- Layer-wise relevance propagation (LRP) [Yeom 2021]
  - Uses a novel criterion, layer-wise relevance propagation, to select weights for structured pruning.

Design of NeuroGRS

NeuroGRS was designed to derive compact DNN models for neural decoding systems.
It can also be applied to generate compact DNN models for other embedded vision applications.

Calcium-imaging-based
neural decoding system:
Overview

NeuroGRS

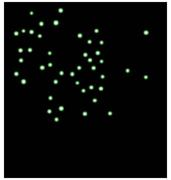
Calcium imaging of brain

Motion correction

Neuron detection

Neural signal extraction

Neural decoding



In real-time: 10 Hz

Prediction of mouse's behavior from analysis of neural signals.

E.g., whether or how fast the mouse is going to move.

Image source:
https://www.nature.com/articles/npp2014206,
https://www.youtube.com/watch?v=d5zK1RUJCiU&ab_channel=MocomiKids.

UNIVERSITY OF MARYLAND

Overview of NeuroGRS

- GRS stands for Greedy inter-layer order with Random Selection of intra-layer units.
- Combines pruning and architecture search with an emphasis on structured pruning.
- Takes into consideration both the model architecture and trained weights.
- Suitable for further compressing small DNN models for optimized embedded implementation.
- Accompanied by a dataflow-based inference system for efficient inference.

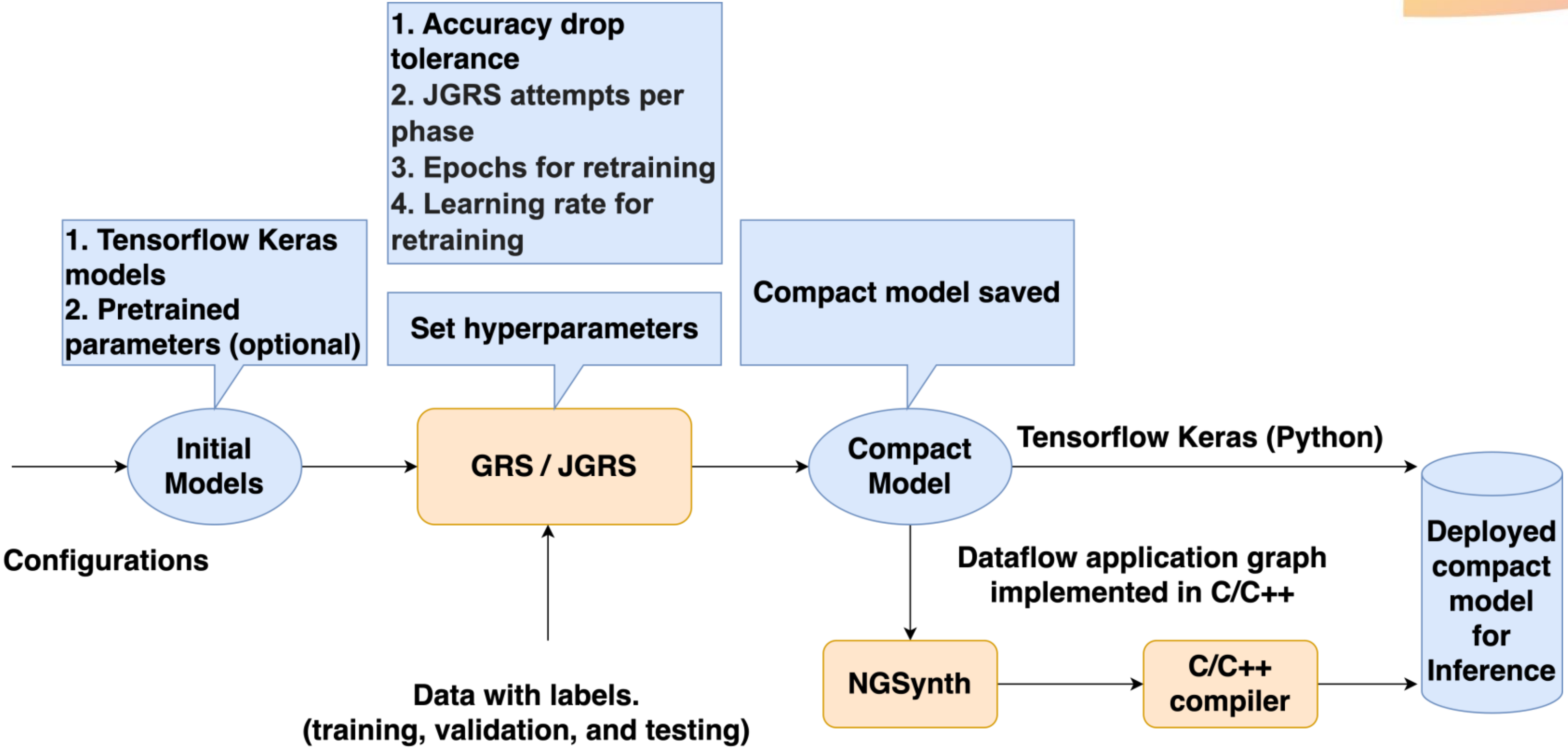
Foundational Findings Applied in NeuroGRS

- Structures determine performance for shallow DNNs; learned weights can be retrained from scratch [Liu 2018] [Frankle 2018].
  - This finding is especially relevant for embedded vision, where shallow DNNs may be preferable due to resource constraints.
- Using a large compression rate (number of removed neurons or connections) without retraining can significantly degrade inference accuracy [Li 2016] [Hu 2016].

How to Use the NeuroGRS Software Package

- Specify initial CNN structure in Keras Tensorflow format.
  - If pretrained, load pretrained weights. GRS can either train first and then prune or perform direct-prune from pretrained model.
- Provide training, validation, and testing data sets.
- Configure hyperparameters.
- Run NeuroGRS to execute the pruning process.
- Output → compact model implemented in Python/C/C++, suitable for inference on embedded platforms

Using NeuroGRS (Continued)



GRS Method (1)

Dataflow graph for NeuroGRS:

M_i: an overparameterized DNN model candidate

$P(M_i)$: a pruned DNN model candidate

v_i: a selection criterion

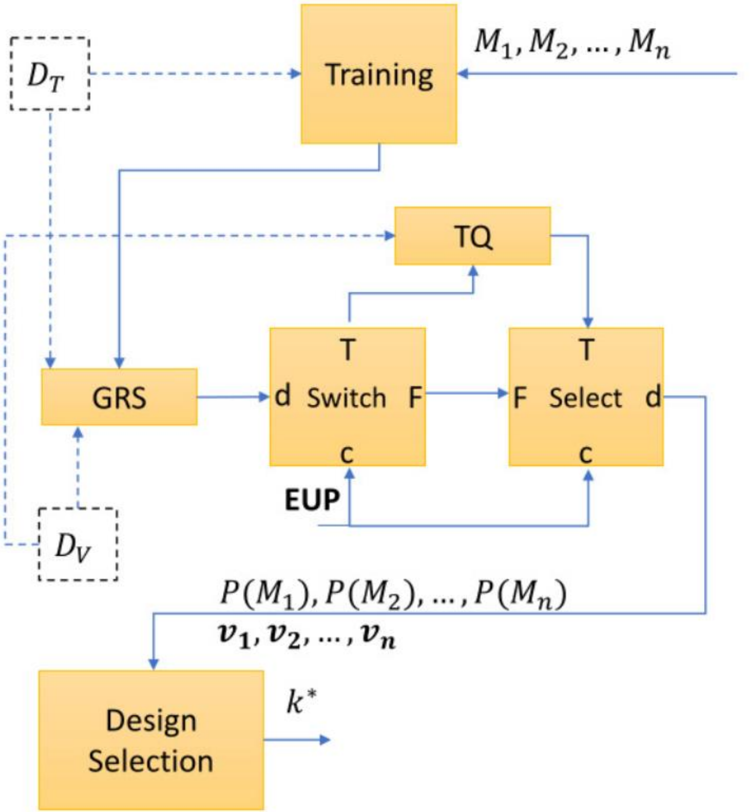
k: the final selected compact model(s)

EUP: Enable Unstructured Pruning

TQ: Thresholding weight connections, Quantization

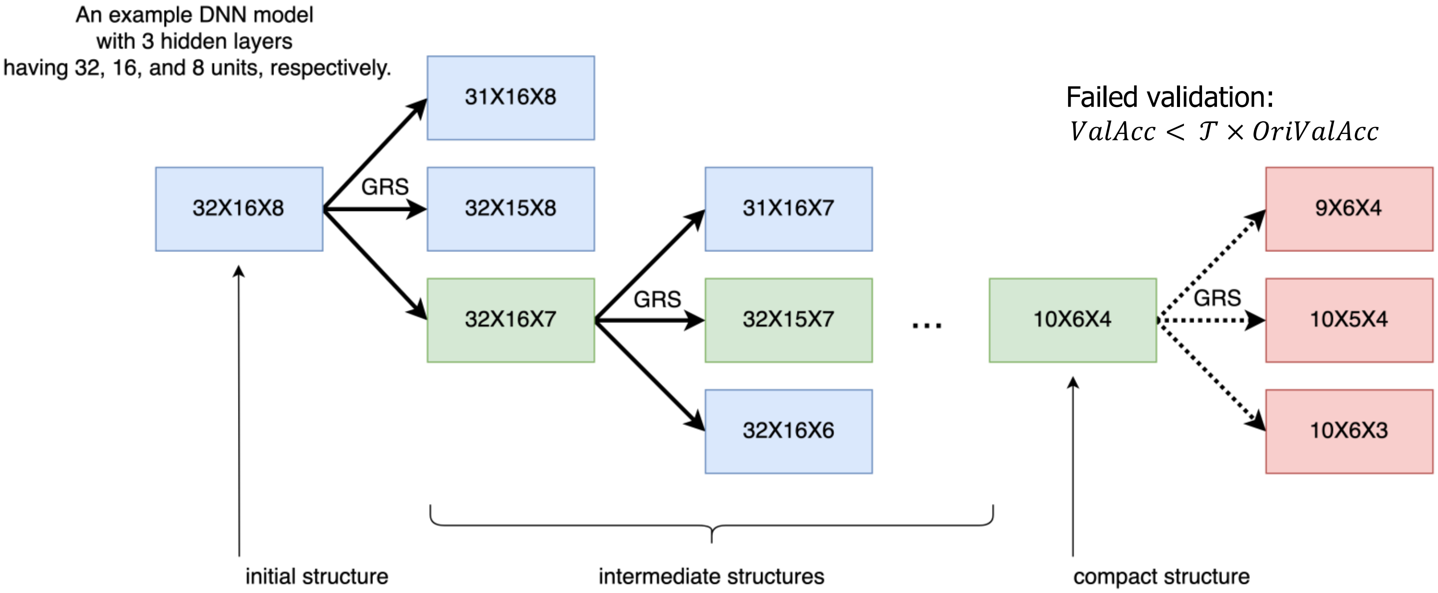
D_T: Training dataset

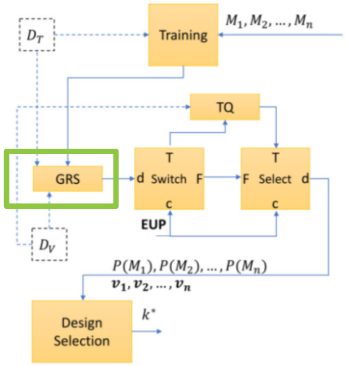
D_V: Validation dataset



GRS Method (2)

GRS: **Greedy inter-layer** order with **Random Selection** of **intra-layer** units



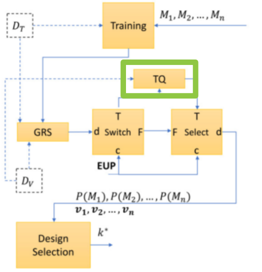


- $ValAcc$: validation accuracy
- $OriValAcc$: validation accuracy of the initial structure
- $\mathcal{T}$: tolerance of accuracy drop

[Wu 2022] X. Wu, D.-T. Lin, R. Chen, and S. Bhattacharyya. Learning compact DNN models for behavior prediction from calcium imaging of neural activity. Journal of Signal Processing Systems, 94:455-472, 2022.

GRS Method (3)

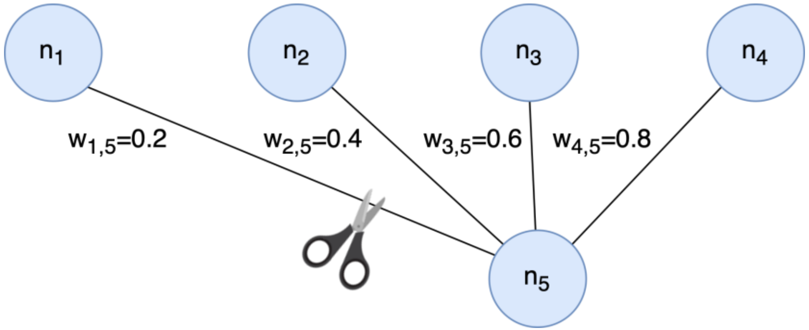
Two state-of-the-art unstructured pruning methods [Han 2015]:



T: Cut weight connections having relatively low magnitudes.

Q: weight quantization.

T:



- $ThreshInit = 0.3$
- $ThreshStep = 0.1$

- Iteratively increase the threshold until accuracy falls below $\mathcal{T} \times OriValAcc$

Q:

w_1=0.998761	w_2=0.132194	w_3=0.000324

Q iterate

w_{1q}=0.999	w_{2q}=0.132	w_{3q}=0.000

Q iterate

w_{1q}=1	w_{2q}=0	w_{3q}=0

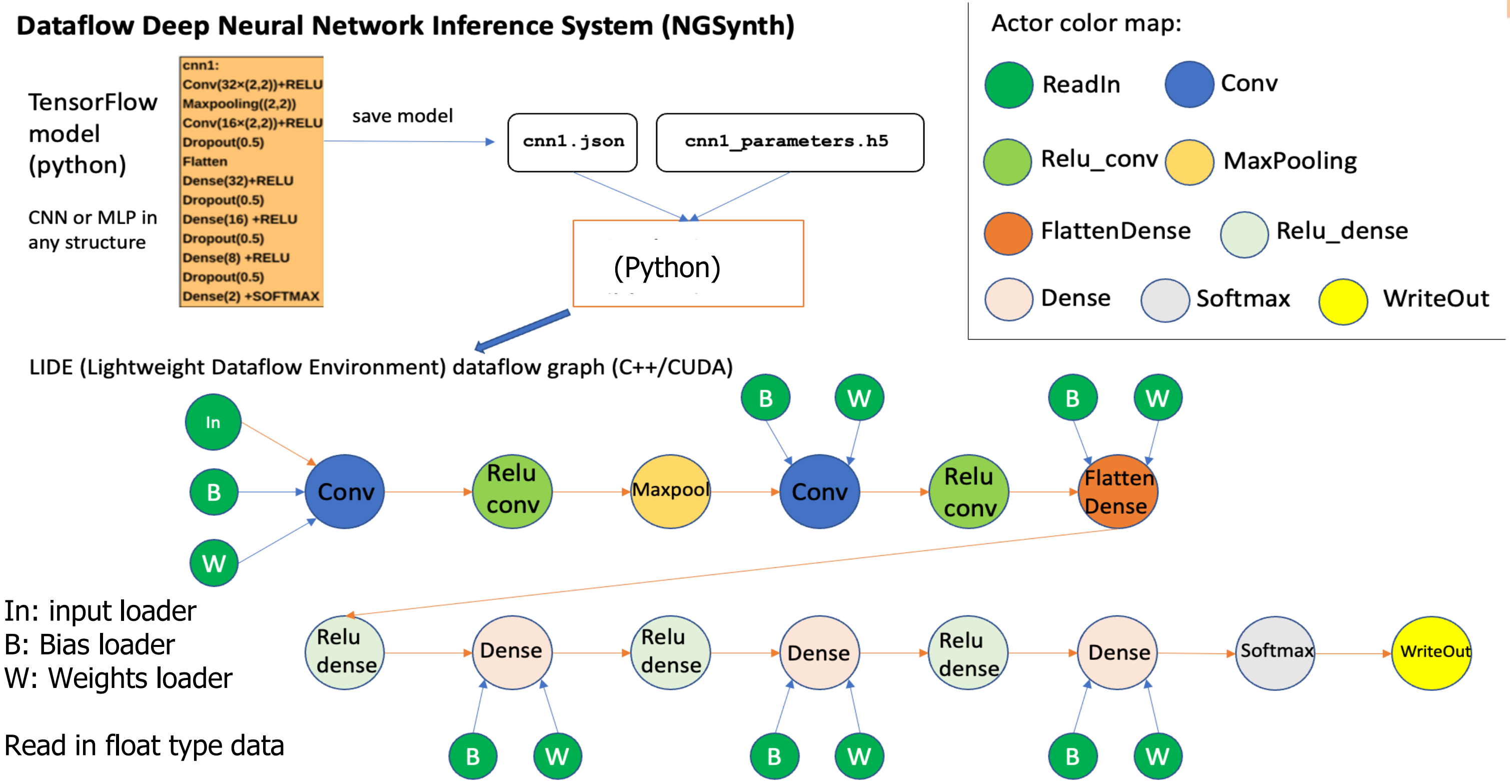
- Iteratively decrease the number of digits until accuracy falls below $\mathcal{T} \times OriValAcc$

UNIVERSITY OF MARYLAND

GRS Method (4)

embedded VISION SUMMIT

Dataflow Deep Neural Network Inference System (NGSynth)



UNIVERSITY OF MARYLAND

Neural Network Models for Evaluation

Initial models in different types and structures:

nn1:
Dense(**32**)+RELU
Dropout(0.5)
Dense(**16**)+RELU
Dropout(0.5)
Dense(**8**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

nn2:
Dense(**32**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

cnn1:
Conv(**32**x(2,2))+RELU
Maxpooling((2,2))
Conv(**16**x(2,2))+RELU
Dropout(0.5)
Flatten
Dense(**32**)+RELU
Dropout(0.5)
Dense(**16**)+RELU
Dropout(0.5)
Dense(**8**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

cnn2:
Conv(**32**x(2,2))+RELU
Maxpooling((2,2))
Conv(**16**x(2,2))+RELU
Dropout(0.5)
Flatten
Dense(**32**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

NeuroGRS Experiments (1)

embedded VISION SUMMIT

Experiment design:

- Investigate whether intermediate sub-structures impact the overall pruning result.
  - Compare GRS with RRS.
    - RRS = **Random inter-layer** order and Random Selection of intra-layer units.
- $\mathcal{T} = 0.985$
- Report average of 4 different models on 9 MSN (Medium Spiny Neuron) datasets with 10 repeated trials each.

Results:

- AL: Test Accuracy Loss, FCI: FLOP Count Improvement, PCI: Parameter Count Improvement.
- Compare GRS with RRS. Metrics: GRS gives X percent more than RRS.

	GRS vs. RRS								
	AL difference			FCI difference			PCI difference		
Model	min	max	avg	min	max	avg	min	max	avg
nn1	-0.32%	1.93%	0.87%	19.96%	67.49%	41.45%	19.91%	67.43%	41.41%
nn2	-0.50%	0.46%	-0.12%	-5.93%	11.25%	2.39%	-5.92%	11.23%	2.40%
cnn1	-2.83%	2.18%	0.74%	47.72%	79.25%	62.92%	47.72%	79.17%	62.87%
cnn2	0.31%	3.51%	1.39%	43.30%	63.75%	56.87%	43.29%	63.63%	56.82%

NeuroGRS Experiments (2)

Experiment design:

- Investigate whether state-of-the-art structured pruning methods for large neural networks are effective in our context.
  - Compare GRS with NWM.
    - NWM = **Natural** inter-layer order and **Weight Magnitude based selection** of intra-layer unit to prune.
    - NWM is representative of other pruning methods that do not consider model structure [Han 2015, Luo 2017].
- $\mathcal{T}$ = 0.985
- Report average of 4 different models on 9 MSN (Medium Spiny Neuron) datasets with 10 repeated trials each.

Results:

- AL: Test Accuracy Loss, FCI: FLOP Count Improvement, PCI: Parameter Count Improvement.
- Compare GRS with NWM. Metrics: GRS gives X percent more than NWM.

	GRS vs. NWM								
	AL difference			FCI difference			PCI difference		
Model	min	max	avg	min	max	avg	min	max	avg
nn1	-0.10%	2.04%	0.78%	8.13%	60.59%	24.18%	8.33%	60.52%	24.22%
nn2	-0.98%	0.50%	0.00%	-11.54%	15.93%	3.03%	-11.54%	15.91%	3.04%
cnn1	-3.48%	1.62%	-0.25%	3.19%	48.33%	21.91%	3.23%	48.27%	21.91%
cnn2	0.00%	3.51%	1.14%	19.13%	50.99%	34.54%	19.08%	50.89%	34.46%

UNIVERSITY OF MARYLAND

NeuroGRS Experiments (3)

Experiment design:

- On 9 MSN (Medium Spiny Neuron) datasets of 3000 frames each.
- 4 different shallow DNN models.
- 10 repeated trials.
- $\mathcal{T}$ of GRS, Pruning Stage T, and Stage Q are set to 0.985, 0.995, and 0.990, respectively.

Results:

- Structured pruning using GRS:

Model	Acc_S (loss%)	FLOPs_S (% of initial)	Params_S (% of initial)
nn1	0.927 (1.03%)	4307 (37.23%)	2177 (37.24%)
nn2	0.931 (0.62%)	5861 (56.98%)	2953 (57.01%)
cnn1	0.9 (0.98%)	8448 (22.5%)	4258 (22.59%)
cnn2	0.917 (1.5%)	9450 (31.33%)	4762 (31.42%)

- Further unstructured pruning with TQ:

Model	Acc_U (loss%)	FLOPs_U (% of initial)	Params_U (% of initial)
nn1	0.923 (1.45%)	2665 (24.69%)	321 (5.72%)
nn2	0.929 (0.79%)	4001 (40.2%)	131 (2.97%)
cnn1	0.895 (1.49%)	7828 (20.7%)	893 (5.37%)
cnn2	0.915 (1.63%)	8828 (28.86%)	2421 (15.69%)

NeuroGRS Experiments (4)

Experiment design:

- With 4 different types of DNN models: use NGSynth to implement their optimized and original forms using LIDE-C, and deploy on a Raspberry Pi Zero W V1.1 platform.
- LIDE-C = Lightweight Dataflow Environment integrated with the C programming language [Lin 2017].
- How much runtime improvement is observed from the compact models compared to their corresponding overparameterized models?

Results:

	Inference runtime		
Model	Original model (ms)	Pruned model (ms)	Improvement
nn1	1.203	0.572	52.44%
nn2	1.074	0.666	37.97%
cnn1	16.277	4.646	71.46%
cnn2	16.139	5.031	68.83%

Jump-GRS Extension

embedded VISION SUMMIT

Overview:

- Identify the **far phase** and the **near phase** of the GRS pruning process.
- Structures impact model performance less in the far phase compared to the near phase.
- This type of phase-based reasoning can be adapted to other pruning methods.
- Develop a "jump mechanism" to help GRS step into the near phase much faster → Much less time (and less carbon footprint) required for pruning.

UNIVERSITY OF MARYLAND

Conclusion

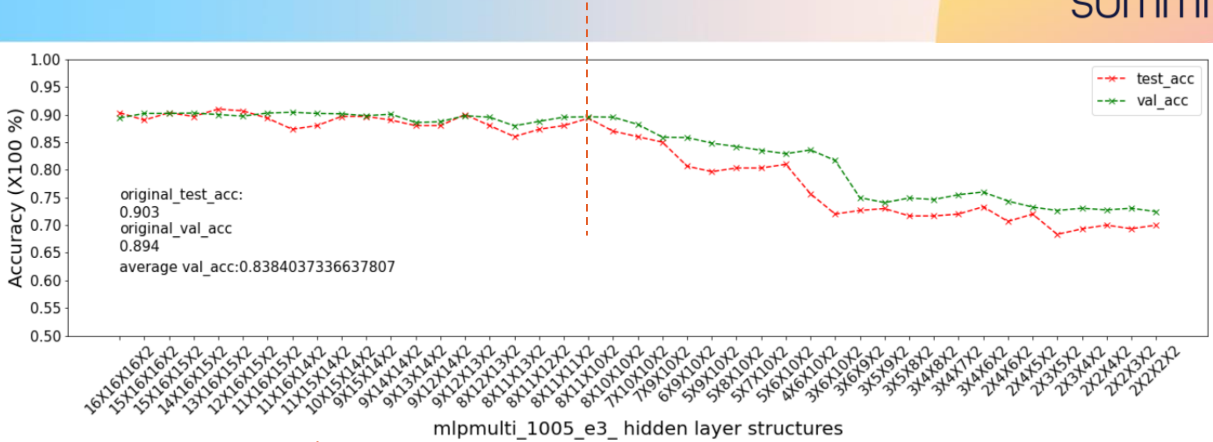
- We have given an overview of pruning and other classes of methods for compressing DNN models.
- We have introduced a new pruning method called Greedy inter-layer order with Random Selection of intra-layer units (GRS).
- We have combined GRS with methods for unstructured pruning to provide a more comprehensive pruning solution.
- We have introduced a software tool, called NeuroGRS, that allows system designers apply the GRS method with a high degree of automation.
- We have introduced concepts of near- and far-phase operation, which are applied in NeuroGRS to greatly improve pruning speed.

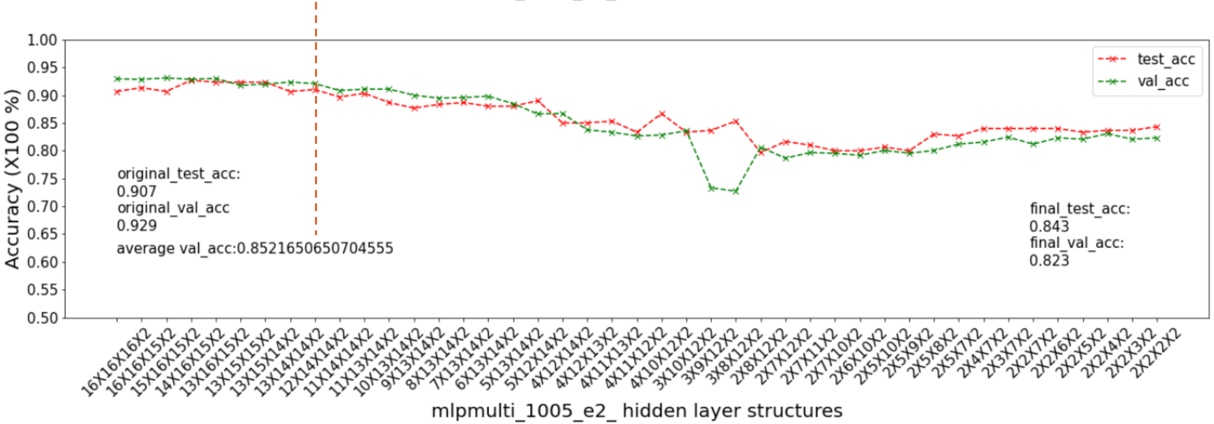
Backup Slides

Jump-GRS Introduction

Demonstrating the pruning phases

- Use RRS (Random inter-layer order, random intra-layer selection of units).
- Retrain and validate all possible intermediate structures 3 times.
- Set $\mathcal{T} = 0.5$ to allow pruning to continue.
- Use an MLP model called "mlpmulti" with a hidden structure of 16X16X16.
- Plot the average validation accuracy of all possible structures with repeats at each pruning step.

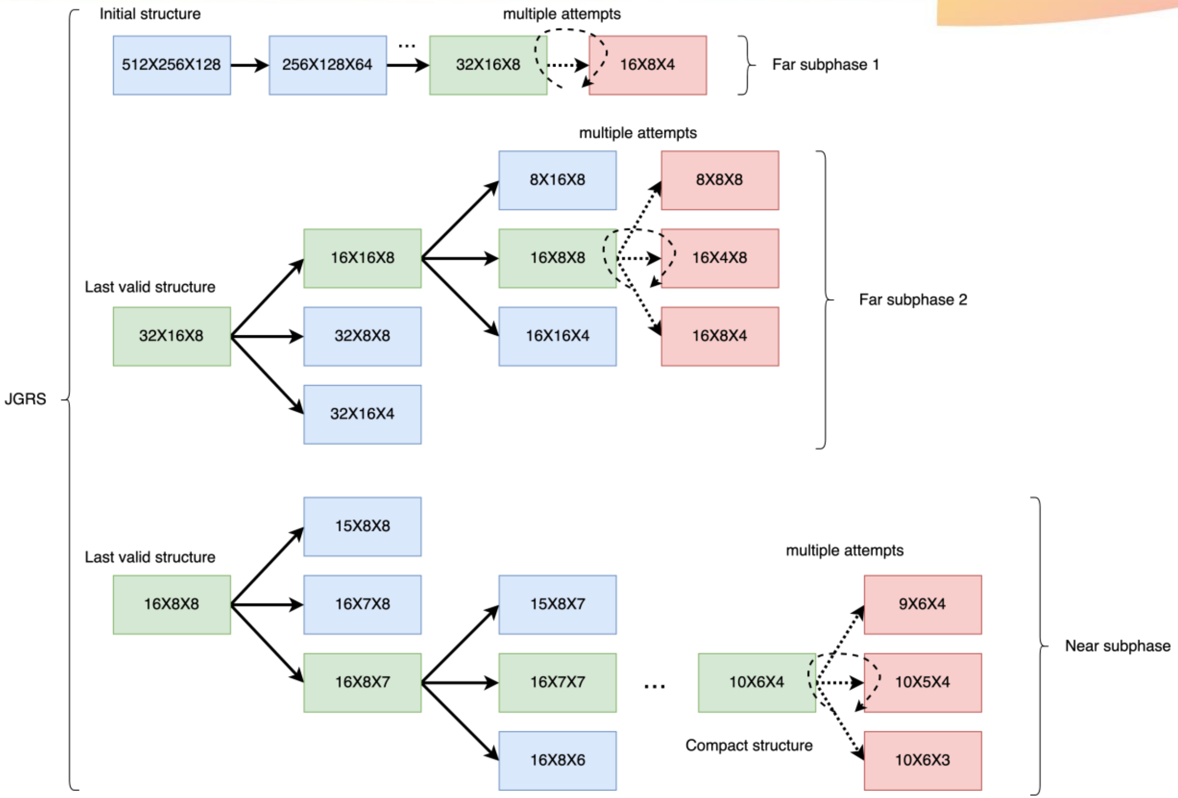




Jump-GRS Method

JGRS algorithm:

- Multiple attempts are used to exploit randomization in the algorithm. The best result across all attempts is taken.
- The last valid structure, after all attempts, will be used as the initial structure for the next phase.
- Each attempt can be regarded as an examination of different cut-off artificial node sets.
- The three phases of GRS have different compression rates.
- JGRS reduces the compression rate as it goes from one subphase/phase to the next.
- A structure fails if its validation acc. becomes lower than $\mathcal{T} \times OriValAcc$.



Evaluation of Jump-GRS Method

Initial DNN models used in NeuroGRS:

nn1:
Dense(**32**)+RELU
Dropout(0.5)
Dense(**16**)+RELU
Dropout(0.5)
Dense(**8**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

nn2:
Dense(**32**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

cnn1:
Conv(**32**x(2,2))+RELU
Maxpooling((2,2))
Conv(**16**x(2,2))+RELU
Dropout(0.5)
Flatten
Dense(**32**)+RELU
Dropout(0.5)
Dense(**16**)+RELU
Dropout(0.5)
Dense(**8**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

cnn2:
Conv(**32**x(2,2))+RELU
Maxpooling((2,2))
Conv(**16**x(2,2))+RELU
Dropout(0.5)
Flatten
Dense(**32**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

Scaled DNN models:

nn1_scaled:
Dense(**512**)+RELU
Dropout(0.5)
Dense(**256**)+RELU
Dropout(0.5)
Dense(**128**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

nn2_scaled:
Dense(**512**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

cnn1_scaled:
Conv(**512**x(2,2))+RELU
Maxpooling((2,2))
Conv(**256**x(2,2))+RELU
Dropout(0.5)
Flatten
Dense(**512**)+RELU
Dropout(0.5)
Dense(**256**)+RELU
Dropout(0.5)
Dense(**128**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

cnn2_scaled:
Conv(**512**x(2,2))+RELU
Maxpooling((2,2))
Conv(**256**x(2,2))+RELU
Dropout(0.5)
Flatten
Dense(**512**)+RELU
Dropout(0.5)
Dense(2)+SOFTMAX

Jump-GRS Experiments (1)

JGRS and GRS Comparison

- 18 datasets (MSN) of 3000 frames.
- Report the average of 10 repeated trials
  - on all MSN datasets.
- $\mathcal{T}$ for both GRS and JGRS is 0.985
- Attempts:
  - Far subphase 1 = 3
  - Far subphase 2 = 3
  - GRS = 3

JGRS vs GRS results:

	JGRS vs GRS											
	AL_difference			FCI_difference			PCI_difference			Time_ratio		
model	min	max	avg	min	max	avg	min	max	avg	min	max	avg
nn1	-8.35%	11.13%	0.28%	-45.47%	93.58%	13.05%	-45.10%	93.51%	13.03%	0.64	23.61	7.95
nn2	-11.87%	9.22%	0.65%	-40.58%	87.39%	19.75%	-40.55%	87.29%	19.72%	0.33	8.11	2.09
cnn1	-14.35%	11.65%	0.00%	-31.40%	93.04%	11.37%	-31.24%	92.98%	11.35%	0.35	18.92	5.58
cnn2	-14.46%	10.87%	0.00%	-50.23%	84.09%	8.78%	-50.20%	83.76%	8.76%	0.3	10.6	4.84

Jump-GRS Experiments (2)

JGRS on larger DNNs

- 18 MSN Datasets of 3000 frames.
- WGEVIA-REAL: 1600 balanced labeled embeddings for two classes of microcircuits.
- $\mathcal{T}$ = 0.985
- Pruning time is reported in *seconds* using a Core i7-2600K CPU with a GeForce GTX 1080 GPU.

GRS on MSN dataset

model	TestAcc_S(lost%)	ValAcc_S(lost%)	FLOPs_S(% of initial)	Paras_S(% of initial)	prune_time
nn1	0.889(1.82%)	0.912(0.65%)	3801(40.86%)	1925(40.87%)	922.9
nn2	0.893(1.61%)	0.911(0.86%)	4361(56.16%)	2203(56.21%)	115.7
cnn1	0.865(2.38%)	0.88(0.46%)	7208(27.34%)	3644(27.47%)	5933.6
cnn2	0.868(2.24%)	0.893(0.55%)	5514(26.06%)	2792(26.2%)	2329.1

JGRS on MSN dataset

model	TestAcc_S(lost%)	ValAcc_S(lost%)	FLOPs_S(% of initial)	Paras_S(% of initial)	prune_time
nn1_scaled	0.904(1.94%)	0.927(0.73%)	5237(1.18%)	2651(1.19%)	346.8
nn2_scaled	0.904(1.99%)	0.926(0.87%)	5665(5.16%)	2863(5.17%)	138.8
cnn1_scaled	0.883(2.90%)	0.907(0.84%)	54926(1.32%)	27579(1.33%)	3064.9
cnn2_scaled	0.887(2.39%)	0.911(0.80%)	33275(1.54%)	16726(1.55%)	1016.8

JGRS on WGEVIA-REAL dataset

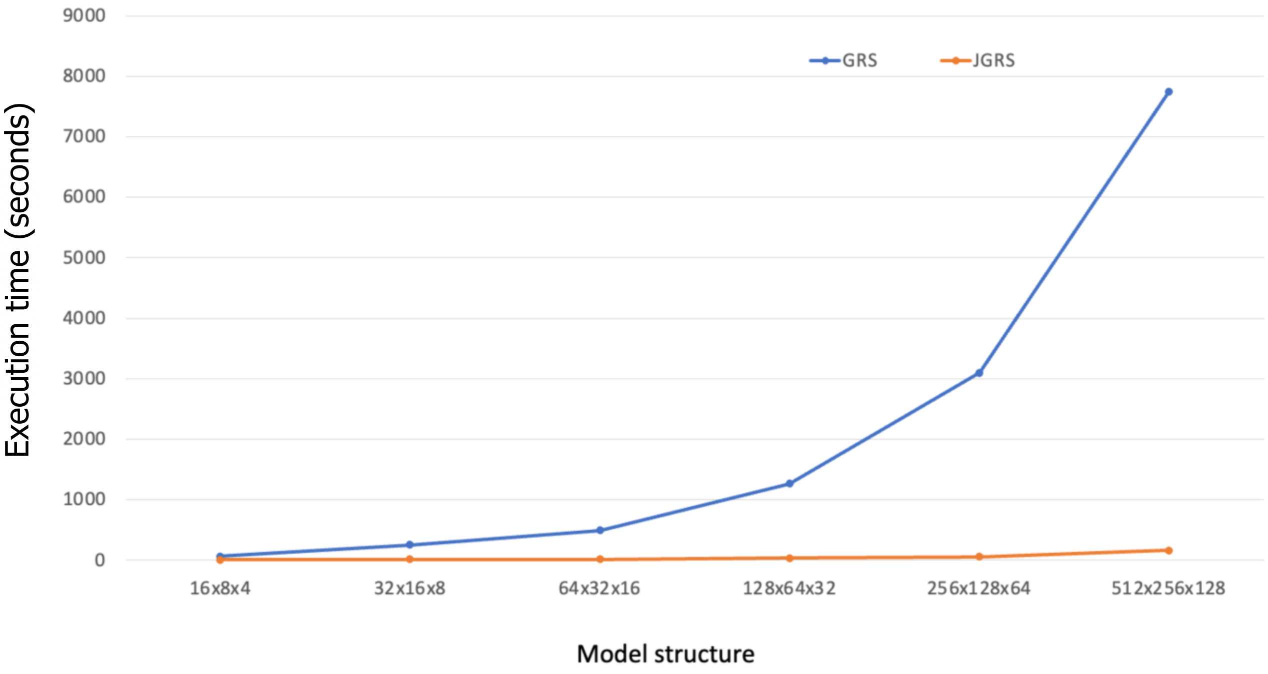
model	TestAcc_S(lost%)	ValAcc_S(lost%)	FLOPs_S(% of initial)	Paras_S(% of initial)	prune_time
nn1_scaled	0.944(1.95%)	0.968(1.20%)	765(0.19%)	394(0.19%)	161.4
nn2_scaled	0.940(2.53%)	0.966(1.21%)	1632(1.94%)	830(1.95%)	94.6
cnn1_scaled	0.954(1.03%)	0.968(0.96%)	2841(0.08%)	1452(0.08%)	1005.5
cnn2_scaled	0.948(1.87%)	0.963(1.25%)	600(0.04%)	316(0.04%)	336.9

UNIVERSITY OF MARYLAND

Jump-GRS Experiments (3)

GRS, JGRS runtime trend:

- WGEVIA-REAL dataset
- MLP models with different numbers of nodes in each layer.
- $\mathcal{T}$ = 0.985
- Plot average runtime (seconds) of 10 repeated trials.



References (1)

- [Bhattacharyya 2019] Bhattacharyya, Shuvra S., et al. "Handbook of Signal Processing Systems." (2019).
- [Han 2015] Han, Song, et al. "Learning both weights and connections for efficient neural network." Advances in neural information processing systems 28 (2015).
- [He 2017] He, Yihui, Xiangyu Zhang, and Jian Sun. "Channel pruning for accelerating very deep neural networks." Proceedings of the IEEE international conference on computer vision. 2017.
- [Hu 2016] Hu, Hengyuan, et al. "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures." arXiv preprint arXiv:1607.03250 (2016).
- [Li 2019] Li, Chunyue, et al. "Prediction of forelimb reach results from motor cortex activities based on calcium imaging and deep learning." Frontiers in cellular neuroscience 13 (2019): 88.
- [Li 2016] Li, Hao, et al. "Pruning filters for efficient convnets." arXiv preprint arXiv:1608.08710 (2016).
- [Lee 2017] Lee, Yaesop, et al. "Online learning in neural decoding using incremental linear discriminant analysis." 2017 IEEE International conference on cyborg and bionic systems (CBS). IEEE, 2017.
- [Lin 2017] Lin, Shuoxin, et al. "The DSPCAD framework for modeling and synthesis of signal processing systems." Handbook of hardware/software codesign. Springer, Dordrecht, 2017. 1185-1219.
- [Liu 2018] Liu, Zhuang, et al. "Rethinking the value of network pruning." arXiv preprint arXiv:1810.05270 (2018).
- [Frankle 2018] Frankle, Jonathan, and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks." arXiv preprint arXiv:1803.03635 (2018).
- [Luo 2017] Luo, Jian-Hao, Jianxin Wu, and Weiyao Lin. "Thinet: A filter level pruning method for deep neural network compression." Proceedings of the IEEE international conference on computer vision. 2017.
- [Molchanov 2016] Molchanov, Pavlo, et al. "Pruning convolutional neural networks for resource efficient inference." arXiv preprint arXiv:1611.06440 (2016).

References (2)

- [Suau 2018] Suau, Xavier, et al. "Principal filter analysis for guided network compression." arXiv preprint arXiv:1807.10585 2 (2018).
- [Wu 2022]X. Wu, D.-T. Lin, R. Chen, and S. Bhattacharyya. Learning compact DNN models for behavior prediction from calcium imaging of neural activity. Journal of Signal Processing Systems, 94:455-472, 2022.
- [Yeom 2021] Yeom, Seul-Ki, et al. "Pruning by explaining: A novel criterion for deep neural network pruning." Pattern Recognition 115 (2021): 107899.