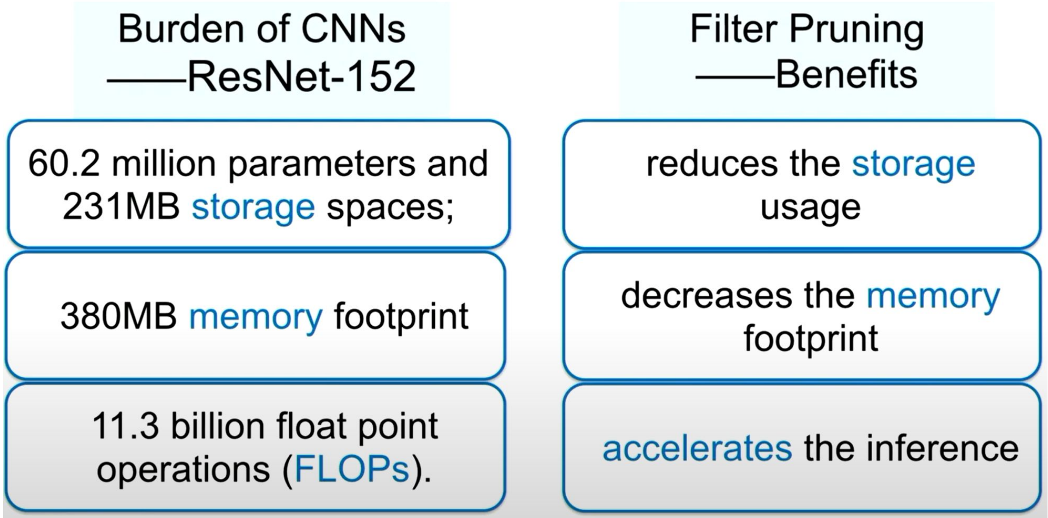
Winning the Lottery Ahead of Time: Efficient Early Network Pruning

John Rachwan, Daniel Zügner, Bertrand Charpentier, Simon Geisler, Morgane Ayle, Stephan Günnemann

CNeRG Reading Group Presentation
(8th June, 2023)

Kiran Purohit
Akash Ghosh

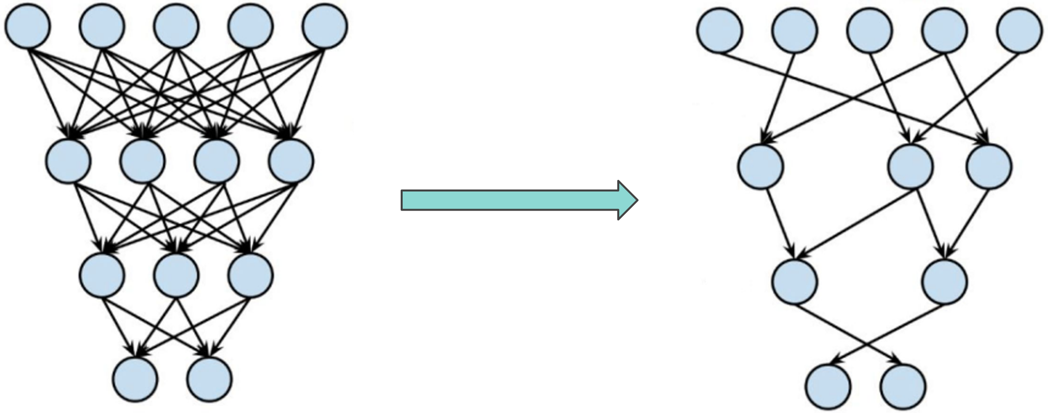
Why pruning is important?



- Neural networks are highly over-parameterized. Pruning helps to remove unnecessary weights and nodes.
- Pruning can help in reducing model memory, training and inference time, cost and carbon emissions while maintaining model performance.

Network Pruning

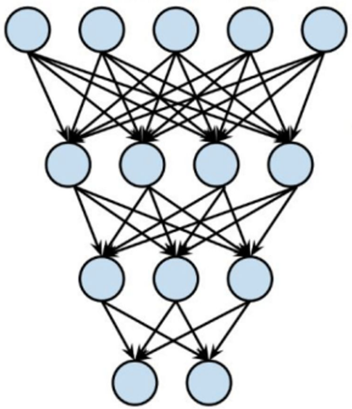
Given a pre-trained network Φ(.), the goal is to compress the network while maintaining the high performance as much as possible by removing the unnecessary parameters.



Pre-trained original network Φ(.)

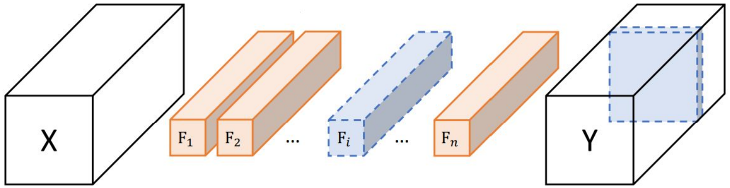
Final pruned network Φ’(.)

Network Pruning



Unstructured Pruning

- ❖ Pruning applied to early DNN
- ❖ Check the importance of each weight or node
- ❖ Practical acceleration could not be achieved



Structured Pruning

- ❖ Widely used for modern CNNs
- ❖ Remove the entire filter or channel at once
- ❖ Helps the practical acceleration of the network

Existing Methods

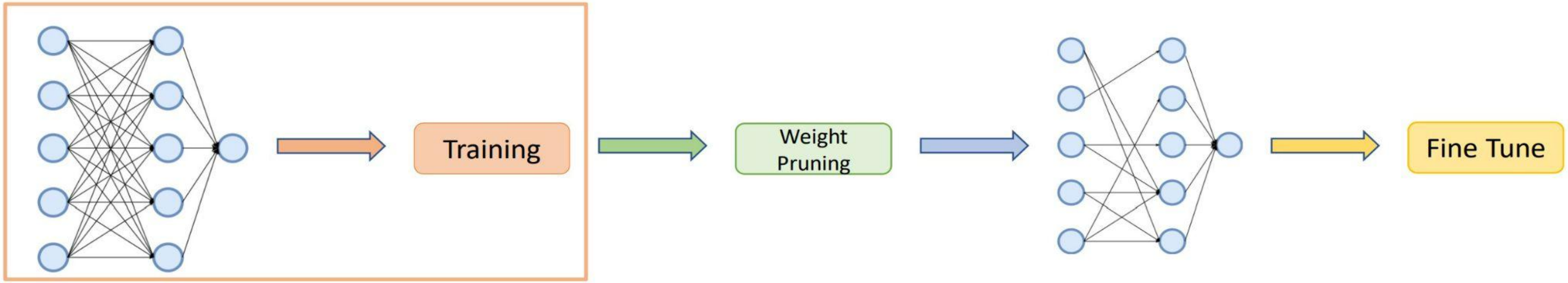
Pruned models can be extracted from two ways:

- From a Pre-trained network
- within the original randomly initialized dense model

Existing Methods

❖ **From a Pre-trained network**

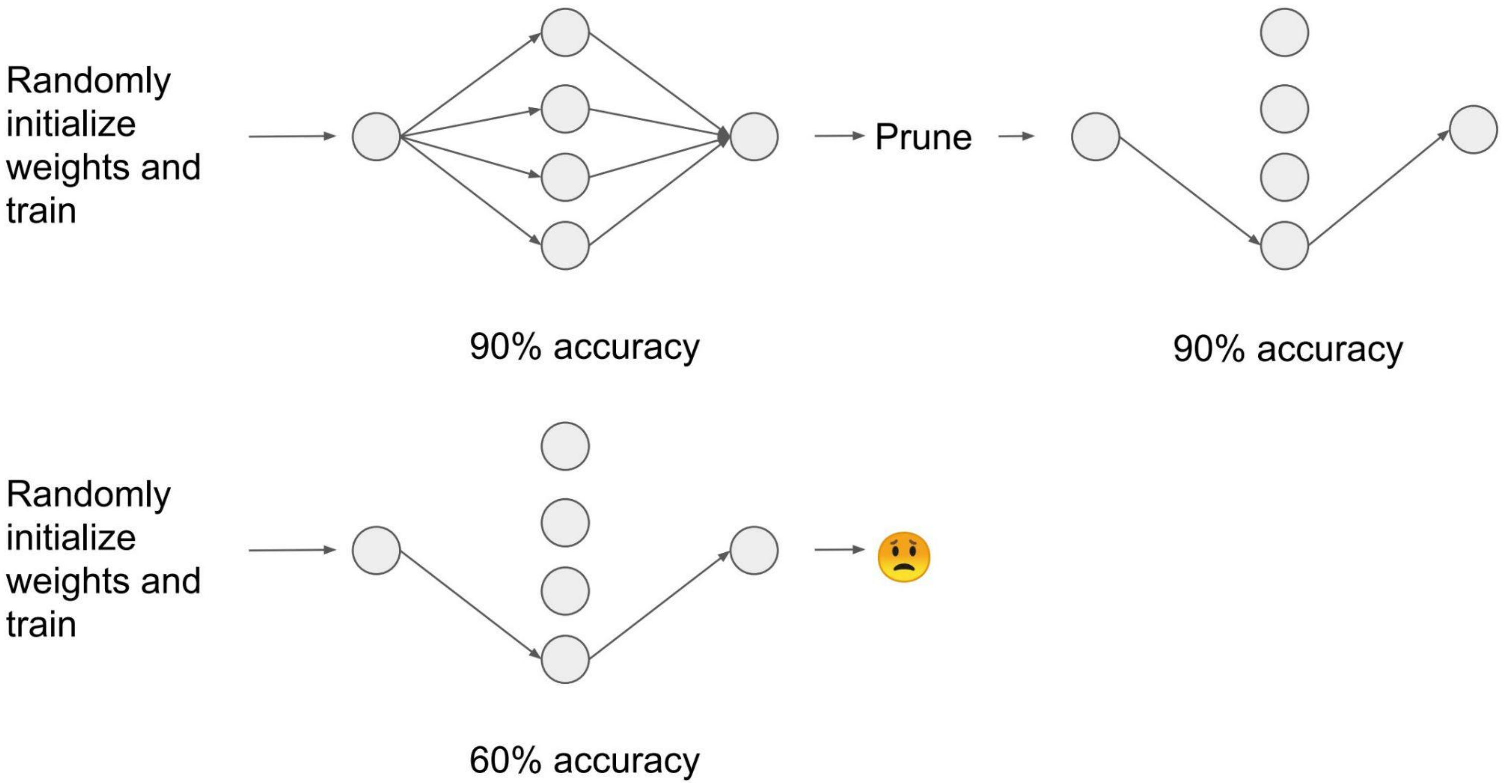
1. Train the model to convergence.
2. Prune the individual weights of the pre-trained model.
3. Fine-tune the sparse model to convergence.



Drawbacks:

- Training phase is as expensive as training the Dense model
- Weight pruning does not practically make weight matrices smaller

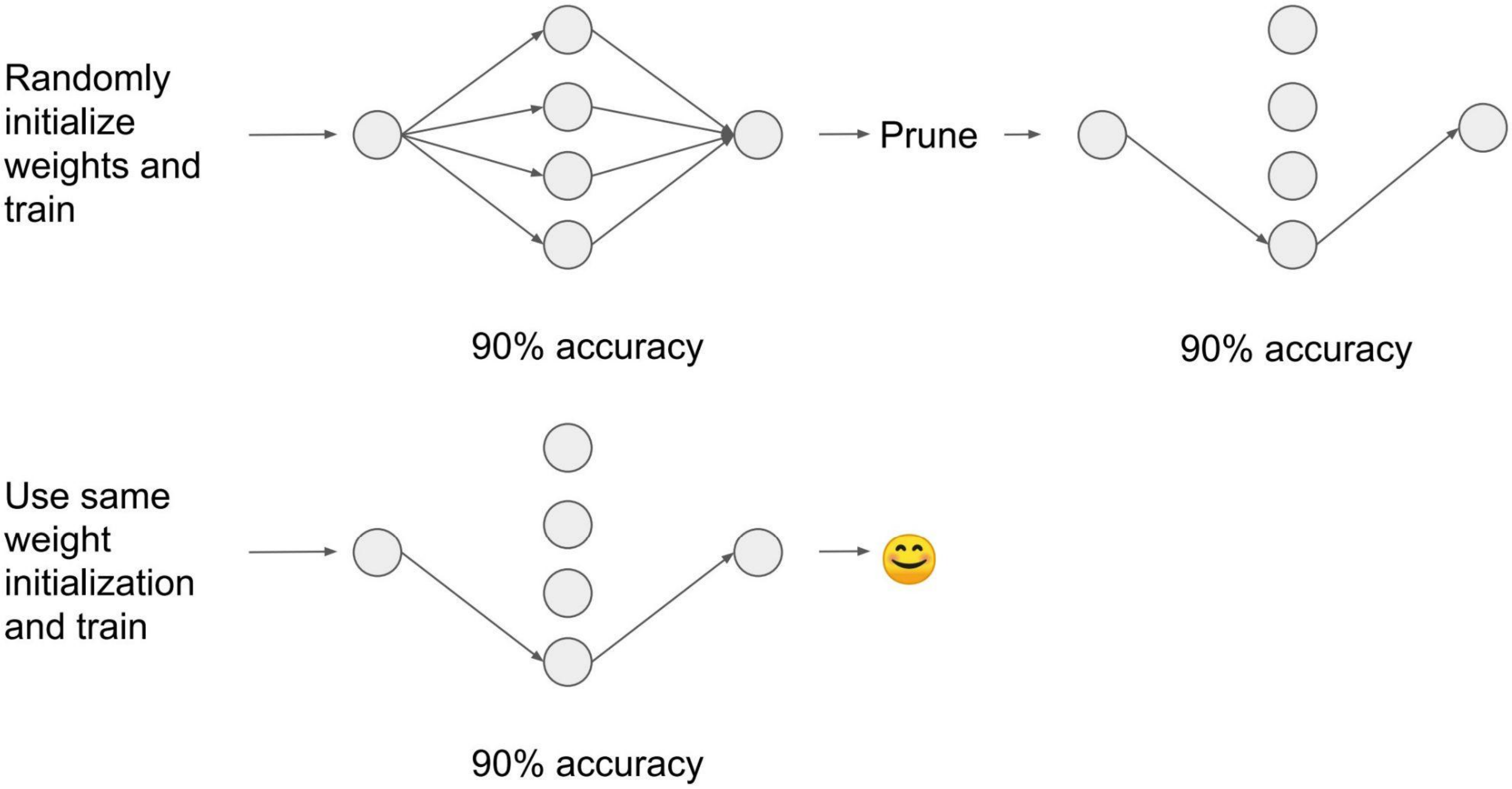
❖ **Within the original randomly initialized dense model**



The Lottery Ticket Hypothesis

A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

The Lottery Ticket Hypothesis



Why the term Lottery?

- ➢ If you want to win the lottery, just buy a lot of tickets and some will likely win.
- ➢ Buying a lot of tickets = having an overparameterized neural network for your task.
- ➢ Winning the lottery = training a network with high accuracy.
- ➢ **Winning ticket** = pruned subnetwork which achieves high accuracy.

Identifying Winning Tickets

One-shot pruning

1. Randomly initialize a neural network with initial parameters θ_0
2. Train the network for j iterations, arriving at parameters θ_j
3. Prune p% of parameters in θ_j with lowest magnitude (set them to 0)
4. Reset the remaining parameters to their original random initialization in θ_0 , creating the winning ticket

Iterative pruning

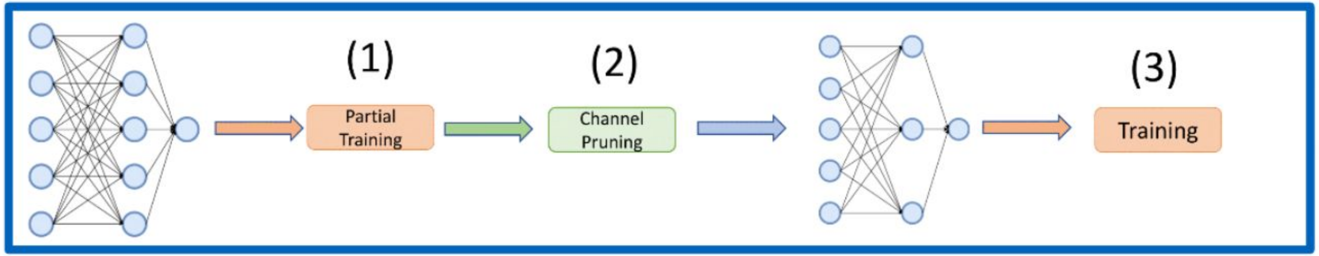
- Iteratively repeat the one-shot pruning process (from step 2)
- Yields smaller networks than one-shot pruning

Limitations

- The process of finding these sparse models using iterative pruning is **very expensive**.
  - Hard to study larger datasets like ImageNet
- Unstructured pruning does not provide benefits in terms of **GPU memory**, **training time**, or **carbon emissions**.

Our Method: EarlyCroP

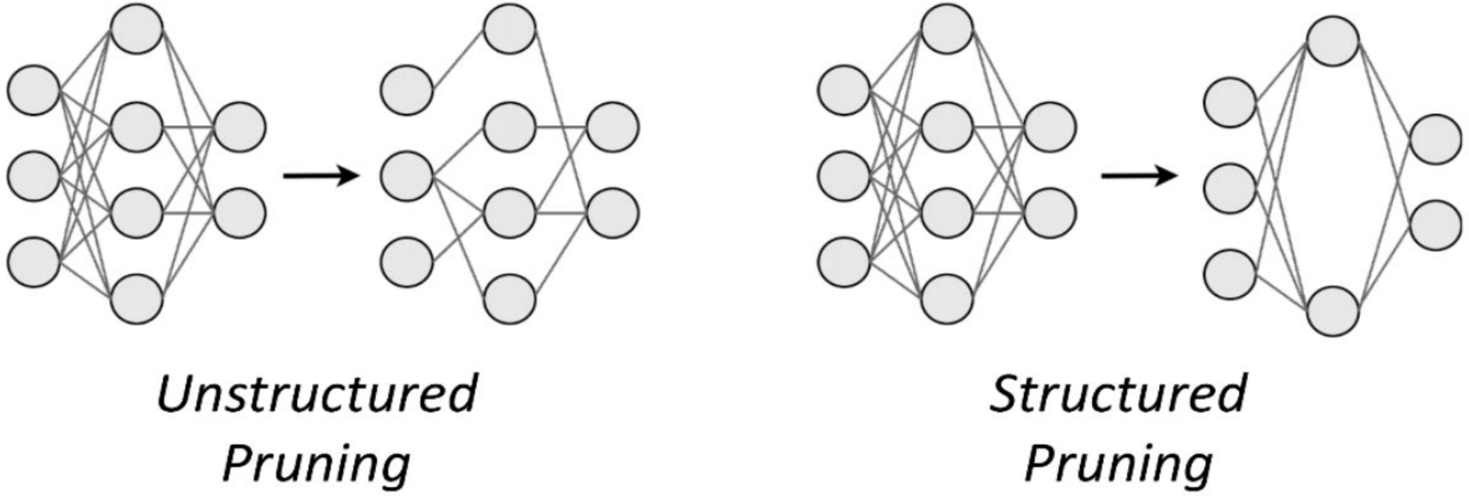
1. Train the model for few epochs
2. Prune the structured channels of the slightly pre-trained model
3. Train the model to convergence.



+ Training is improved: We operate on a slightly trained network
+ Inference is improved: We perform structured pruning

- **Why to prune ?** We gain performance improvements by pruning *structure* layer channels instead of *individual* weights.
- **How to prune ?** We score and remove channels with the goal of preserving the *Gradient Flow* and *Neural Tangent Kernel*.
- **When to prune ?** We prune the network during training when we detect the transition from *rich active* to *lazy kernel regime.*

Why to Prune?



Pruning leads to a sparse model that provides improvements in terms of **time**, **memory** and **carbon emissions**.

How to Prune?

Background:

Neural Tangent Kernel (NTK):

$NTK(\theta) = g_Y(\boldsymbol{\Theta}_t)^T g_Y(\boldsymbol{\Theta}_t)$

where $g_Y(\boldsymbol{\Theta}_t)$ denotes the gradient of the model prediction Y with respect to the model parameters $\boldsymbol{\Theta}_t$.

- ➢ It describes the dynamics of the network's prediction during training.

Gradient flow (GF):

$GF(\theta) = g_L(\boldsymbol{\Theta}_t)^T g_L(\boldsymbol{\Theta}_t)$

where $g_L(\boldsymbol{\Theta}_t)$ denotes the gradient of the model loss L with respect to the model parameters $\boldsymbol{\Theta}_t$.

- ➢ It describes the dynamics of the gradient norm during training.

How to Prune?

A **weight importance score** is assigned to each weight, ordering is done based on the weights that least affect the GF. Here, $H_L(\boldsymbol{\Theta}_t)$ is the model's Hessian at time t.

$$I(\boldsymbol{\Theta}_t) = |\boldsymbol{\Theta}_t^T H_L(\boldsymbol{\Theta}_t) g_L(\boldsymbol{\Theta}_t)|$$

ρ% of the parameters are removed (pruned) with the lowest scores.

The following equation shows that preserving GF is equivalent to preserving NTK.

(Preserving GF <-> Preserving NTK)

$$\begin{aligned} \text{GF} &= g_L(\boldsymbol{\Theta}_t)^T g_L(\boldsymbol{\Theta}_t) \\ &= g_L(Y)^T g_Y(\boldsymbol{\Theta}_t)^T g_Y(\boldsymbol{\Theta}_t) g_L(Y) \\ &= g_L(Y)^T \, \text{NTK} \, g_L(Y) \end{aligned}$$

Hence Pruning the weights with the lowest importance score preserves both GF and NTK.

When to Prune?

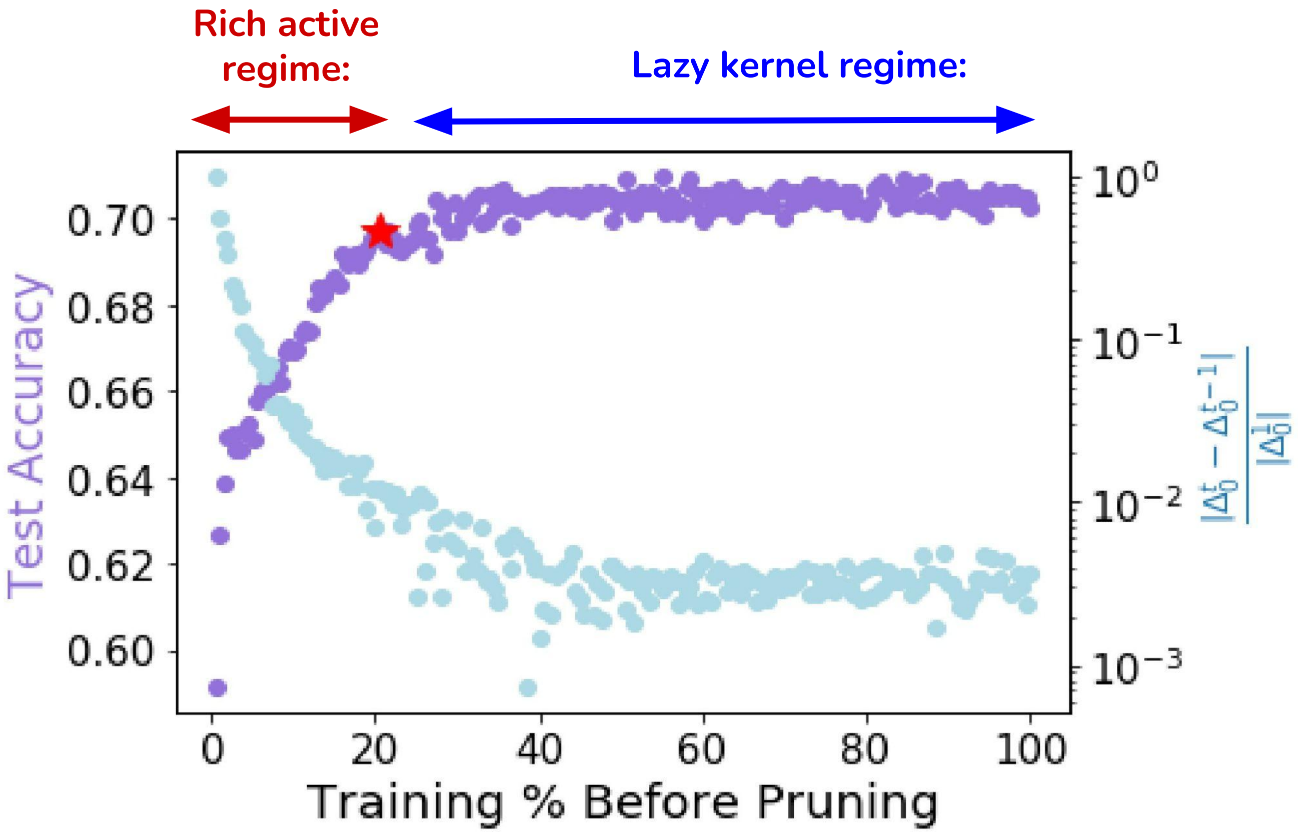
Two model training dynamics are considered:

Rich active regime: Weights are changing relatively quickly.

Lazy kernel regime: Weights are changing slightly. In this area, the NTK is constant and pruning does not affect model accuracy.

EarlyCroP tries to detect the best time for pruning. The timing for pruning is considered best at the transition from rich active regime to lazy kernel regime.

When to Prune?



When to Prune?

We detect the transition ★ from rich to lazy regime with the **pruning time detection score**:

- At every epoch we compute the pruning time score.

  $\Delta^t_0 = || \Theta_t - \Theta_0 ||^2$

- If the difference of the scores at two subsequent epochs relative to the initial score Δ^1_0 is smaller than a defined threshold then we do the pruning.

  $|\Delta^t_0 - \Delta^{t-1}_0| \, / \, |\Delta^1_0| < \text{threshold}$

Experimental Results

Experimental settings

Experiments are performed over four different tasks.

Image classification: The datasets used for image classification are common benchmarks CIFAR10, CIFAR100 and Tiny-Imagenet. The networks used are ResNet18, VGG16, ResNeXt101 32*16d and ResNext-101 32*48d.

Regression: A Fully Convolutional Residual Network is trained on the NYU Depth Estimation Task (Nathan Silberman & Fergus, 2012).

Natural Language Processing (NLP): Pointer Sentinel Mixture Model (Merity et al. 2017) is trained on PTB language modeling dataset (Marcus et al. 1993).

Reinforcement Learning (RL): The FLARE framework (Yu et al. 2018) is used to evaluate a simple 3-layer FCNN on the control game CartPole-v0 (Brockman et al. 2016).

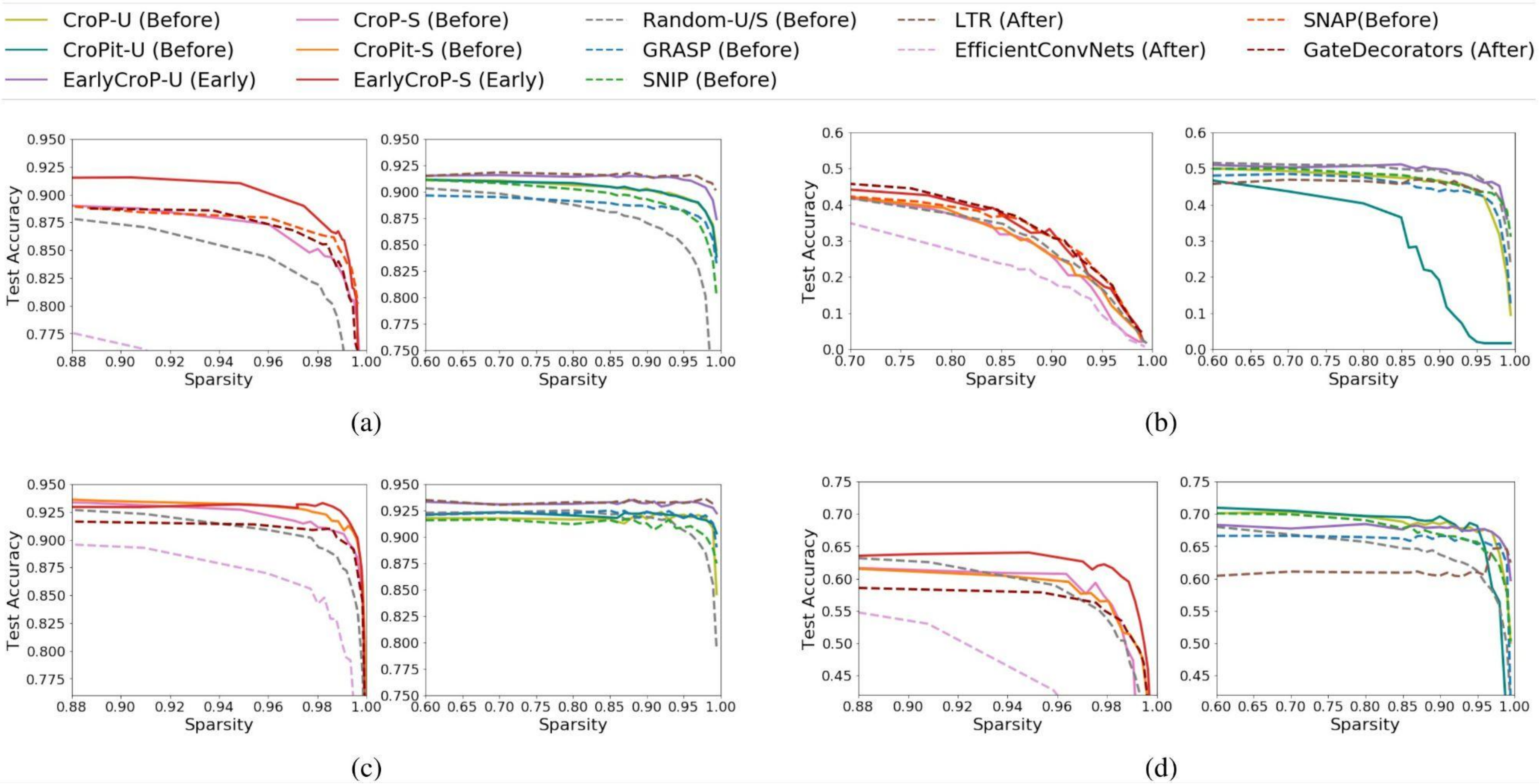


Figure: Structured (Left) and Unstructured (Right) test accuracy for (a) ResNet18/CIFAR10, (b) ResNet18/Tiny-Imagenet, (c) VGG16/CIFAR10 and (d) VGG16/CIFAR100 with increasing weight sparsity.

Table. **Comparison between different pruning criteria on ResNet18/CIFAR10 at 95% sparsity, averaged over 3 runs. ↑/↓ indicate metrics where higher/ lower is better. GPU RAM and Disk correspond to those of the final pruned model.**

	Method	Test accuracy ↑	Weight sparsity	Node sparsity	Training time (h) ↓	Batch time (ms) ↓	GPU RAM (GB) ↓	Disk (MB) ↓	Emissions (g) ↓
	Dense	91.5% ± 0.12	-	-	0.78	109	2.38	398	83
Structured	Random-S	86.3% ± 0.06	93.7%	75.0%	0.68	82	0.62	24.9	38
	SNAP	87.6% ± 0.94	93.6%	72.6%	0.70	81	0.63	25.4	39
	CroP-S	87.5% ± 0.36	93.6%	72.3%	0.67	91	0.63	25.4	43
	CroPit-S	<u>87.8%</u> ± 0.33	95.0%	74.5%	<u>0.52</u>	<u>0.48</u>	0.59	19.6	35
	EarlyBird	84.3% ± 0.32	95.3%	65.0%	**0.48**	72	0.58	19.1	55
	EarlyCroP-S	**91.0%** ± 0.52	95.1%	65.8%	<u>0.52</u>	**66**	0.56	19.2	68
	GateDecorators	87.3% ± 0.09	95.7%	73.7%	0.72	83	0.58	17.2	54
	EfficientConvNets	70.5% ± 0.53	95.9%	79.7%	0.77	83	0.76	25.4	63
Unstructured	Random-U	84.9% ± 0.24	95.0%	-	0.78	<u>102</u>	2.86	12.0	79
	SNIP	88.2% ± 0.57	95.0%	-	0.79	105	2.86	12.0	80
	GRASP	88.4% ± 0.13	95.0%	-	0.79	106	2.84	12.0	81
	CroP-U	87.9% ± 0.16	95.0%	-	<u>0.75</u>	107	2.88	12.0	79
	CroPit-U	89.1% ± 0.24	95.0%	-	0.80	113	2.88	12.0	87
	EarlyCroP-U	<u>91.1%</u> ± 0.23	95.0%	-	**0.74**	**97**	2.86	12.0	83
	LTR	**91.5%** ± 0.26	95.0%	-	1.94	111	2.51	12.0	202

Table. **Comparison between different pruning criteria on VGG16/CIFAR10 at 98% sparsity, averaged over 3 runs. ↑/↓ indicate metrics where higher/ lower is better. GPU RAM and Disk correspond to those of the final pruned model.**

	Method	Test accuracy ↑	Weight sparsity	Node sparsity	Training time (h) ↓	Batch time (ms) ↓	GPU RAM (GB) ↓	Disk (MB) ↓	Emissions (g) ↓
-	Dense	90.2%	-	-	1.82	290	1.02	1720	246
Structured	Random-S	89.3%	98.0%	86.1%	**0.67**	**82**	0.23	33.6	43
Structured	SNAP	89.8%	98.2%	89.0%	0.68	89	0.22	30	55
Structured	CroP-S	91.1%	98.0%	88.0%	0.71	91	0.23	33.6	83
Structured	CroPit-S	92.4%	98.0%	88.0%	0.81	112	0.23	30.4	100
Structured	EarlyBird	85.9%	98%	89 %	0.52	110	0.32	36.2	160
Structured	EarlyCroP-S	**93.0%**	98.0%	89.0%	1.16	112	0.63	36.0	156
Structured	GateDecorators	90.0%	98.0%	87.0%	1.07	111	0.23	37.8	143
Structured	EfficientConvNets	84.2%	98.0%	86.0%	1.66	89	0.64	34.2	209
Unstructured	Random-U	88.5%	98.0%	-	2.03	159	1.22	35.0	247
Unstructured	SNIP	90.1%	98.0%	-	2.02	157	1.22	35.0	248
Unstructured	GRASP	92.0%	98.0%	-	2.03	157	1.23	35.0	249
Unstructured	CroP-U	91.8%	98.0%	-	2.02	157	1.22	35.0	248
Unstructured	CroPit-U	91.6%	98.0%	-	2.02	157	1.22	35.0	249
Unstructured	EarlyCroP-U	93.0%	98.0%	-	**2.01**	157	1.22	35.0	250
Unstructured	LTR	**93.6%**	98.0%	-	4.07	158	1.22	35.0	592

Table. **Comparison between different pruning criteria on VGG16/CIFAR100 at 98% sparsity. ↑/↓ indicate metrics where higher/ lower is better. GPU RAM and Disk correspond to those of the final pruned model.**

	Method	Test accuracy ↑	Weight sparsity	Node sparsity	Training time (h) ↓	Batch time (ms) ↓	GPU RAM (GB) ↓	Disk (MB) ↓	Emissions (g) ↓
-	Dense	62.1%	-	-	0.77	114	1.03	1745	88
Structured	Random-S	53.9%	98.0%	86.0%	**0.59**	53	0.23	35	29
	SNAP	49.3%	98.0%	89.0%	0.67	54	0.16	36	33
	CroP-S	<u>57.4%</u>	98.0%	89.0%	<u>0.61</u>	<u>46</u>	0.23	36	35
	CroPit-S	56.5%	98.1%	89.0%	0.62	**44**	0.23	33	30
	EarlyBird	60.7%	98.0%	89.0%	0.56	68	0.20	36	62
	EarlyCroP-S	**62.2%**	97.9%	88.0%	0.64	69	0.23	36	58
	GateDecorators	55.0%	97.9%	87.0%	<u>0.61</u>	78	0.23	36	68
	EfficientConvNets	29.5%	98.0%	86.0%	0.72	55	0.24	36	83
Unstructured	Random-U	55.8%	98.0%	-	0.74	118	1.23	35	99
	SNIP	61.9%	98.0%	-	0.79	109	1.24	35	90
	GRASP	63.4%	98.0%	-	0.79	113	1.24	35	91
	CroP-U	63.8%	98.0%	-	**0.74**	109	1.23	35	94
	CroPit-U	56.3%	98.0%	-	**0.74**	111	1.23	35	91
	EarlyCroP-U	**65.1%**	98.0%	-	**0.74**	109	1.23	35	91
	LTR	<u>64.7%</u>	98.0%	-	3.44	109	1.28	35	301

Table. **Comparison between different pruning criteria on ResNet18/TinyImageNet at 90% sparsity. ↑/↓ indicate metrics where higher/ lower is better. GPU RAM and Disk correspond to those of the final pruned model.**

	Method	Test accuracy ↑	Weight sparsity	Node sparsity	Training time (h) ↓	Batch time (ms) ↓	GPU RAM (GB) ↓	Disk (MB) ↓	Emissions (g) ↓
-	Dense	51.3%	-	-	7.26	320	3.53	569	882
Structured	Random-S	37.3%	91.2%	80.0%	6.23	289	1.08	51	464
Structured	SNAP	38.3%	90.4%	82.6%	**6.06**	268	0.84	55	514
Structured	CroP-S	<u>39.1</u>%	90.1%	77.7%	6.72	237	1.11	54	615
Structured	CroPit-S	<u>39.1</u>%	91.4%	79.3%	6.66	236	1.08	49	591
Structured	EarlyCroP-S	**39.2**%	90.8%	84.1%	7.03	<u>202</u>	0.25	49	676
Structured	GateDecorators	30.1%	89.2%	91.2%	<u>6.20</u>	**193**	0.87	61	930
Structured	EfficientConvNets	27.7%	91.0%	79.8%	6.60	226	0.22	52	769
Unstructured	Random-U	<u>49.3</u>%	90.0%	-	7.25	351	4.20	57	932
Unstructured	SNIP	46.2%	90.0%	-	7.27	314	4.18	57	854
Unstructured	GRASP	43.7%	90.0%	-	7.27	315	4.22	57	881
Unstructured	CroP-U	46.7%	90.0%	-	7.26	314	4.22	57	877
Unstructured	CroPit-U	19.1%	90.0%	-	7.26	313	4.22	57	890
Unstructured	EarlyCroP-U	**49.8**%	90.0%	-	7.26	314	4.23	57	880
Unstructured	LTR	46.3%	90.0%	-	44.7	603	3.68	57	5540

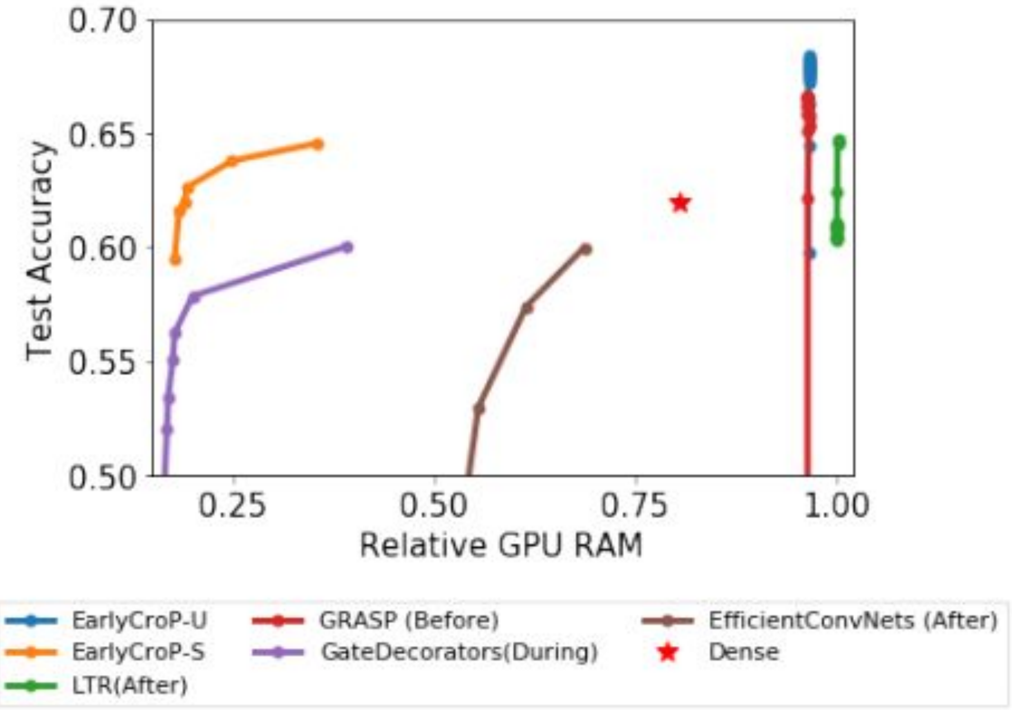


Fig.: **Tradeoff between GPU RAM consumption and test accuracy. EarlyCroP-S gives the best values. Dataset: CIFAR-100, architecture-VGG16**

Table. Comparison of a pruned ResNext101 32x48d (RN48) model and a similar sized dense ResNext101 32x16d (RN16) model (CIFAR10). RN48-S are models pruned with CroP-S.

Model	Test acc.	Weight sparsity	Node sparsity	Epochs	Training time (h)	VRAM (GB)	Emissi-ons (g)
RN48	92.4%	-	-	30	4.60	18.84	634
RN16	92.1%	-	-	30	4.02	3.89	445
RN48-S	**92.5%**	98.5%	89.9%	30	**0.64**	3.56	47
RN48-S	93.2%	98.5%	89.9%	80	2.60	3.56	194

Regression

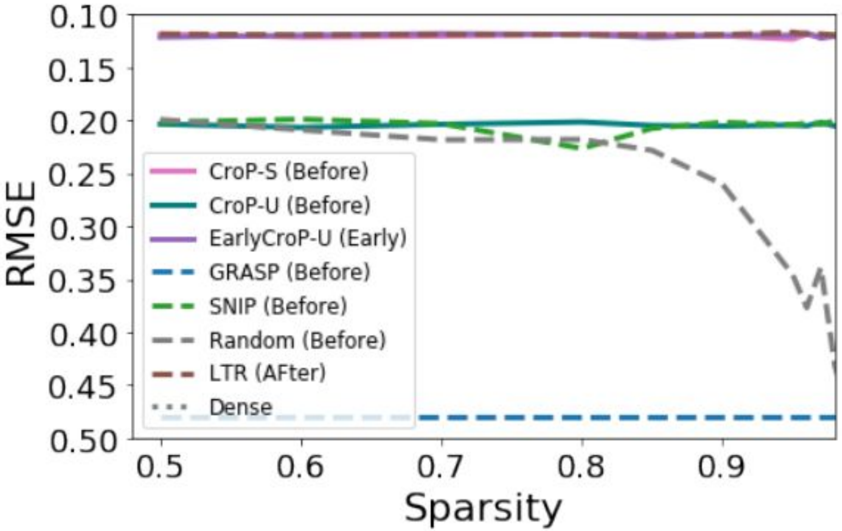


Fig.: Sparse model performance on NYU Depth Estimation task by RMSE

Reinforcement Learning

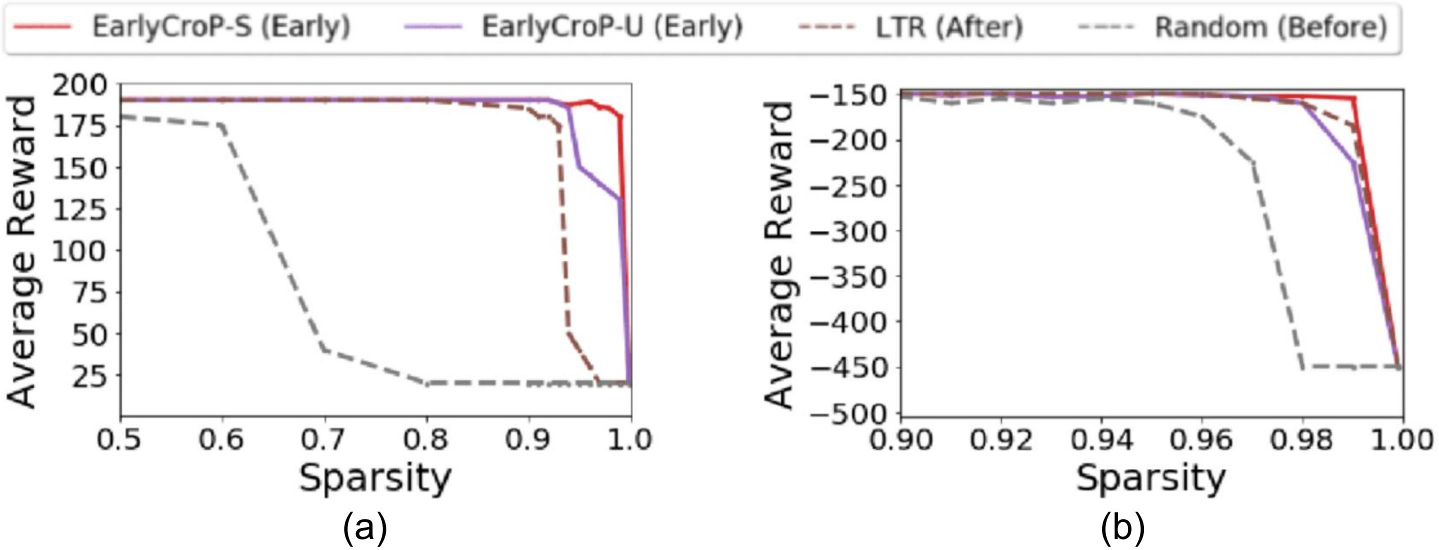


Fig: **Sparse model performance on CartPole control game environments (a) Acrobot-v1 and (b) LunarLander - 2**

Natural Language Processing

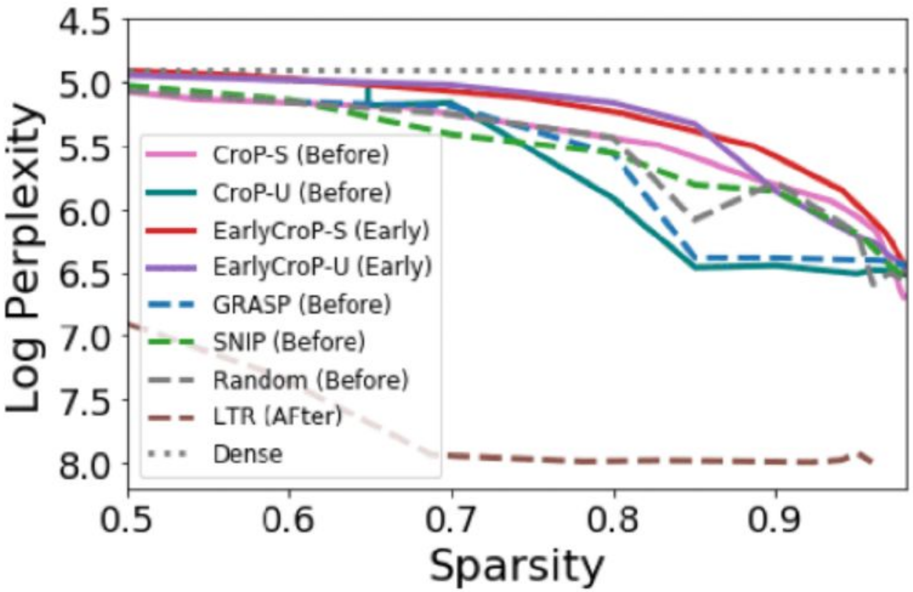


Fig: Sparse model performance on PTB language modelling task in log perplexity.

Conclusion

- ★ EarlyCroP outperforms other pruning methods, providing the best trade-off between test accuracy and efficiency in terms of time, space, and carbon emissions.
- ★ EarlyCroP can train models that do not fit on commodity GPUs by extracting sparse models that preserve the original model's performance.

THANK YOU
FOR
YOUR ATTENTION!!!