

# **ニューラルネットワークを小さくしよう！**

## **～ 宝くじ仮説と枝刈り～**

M2 上野

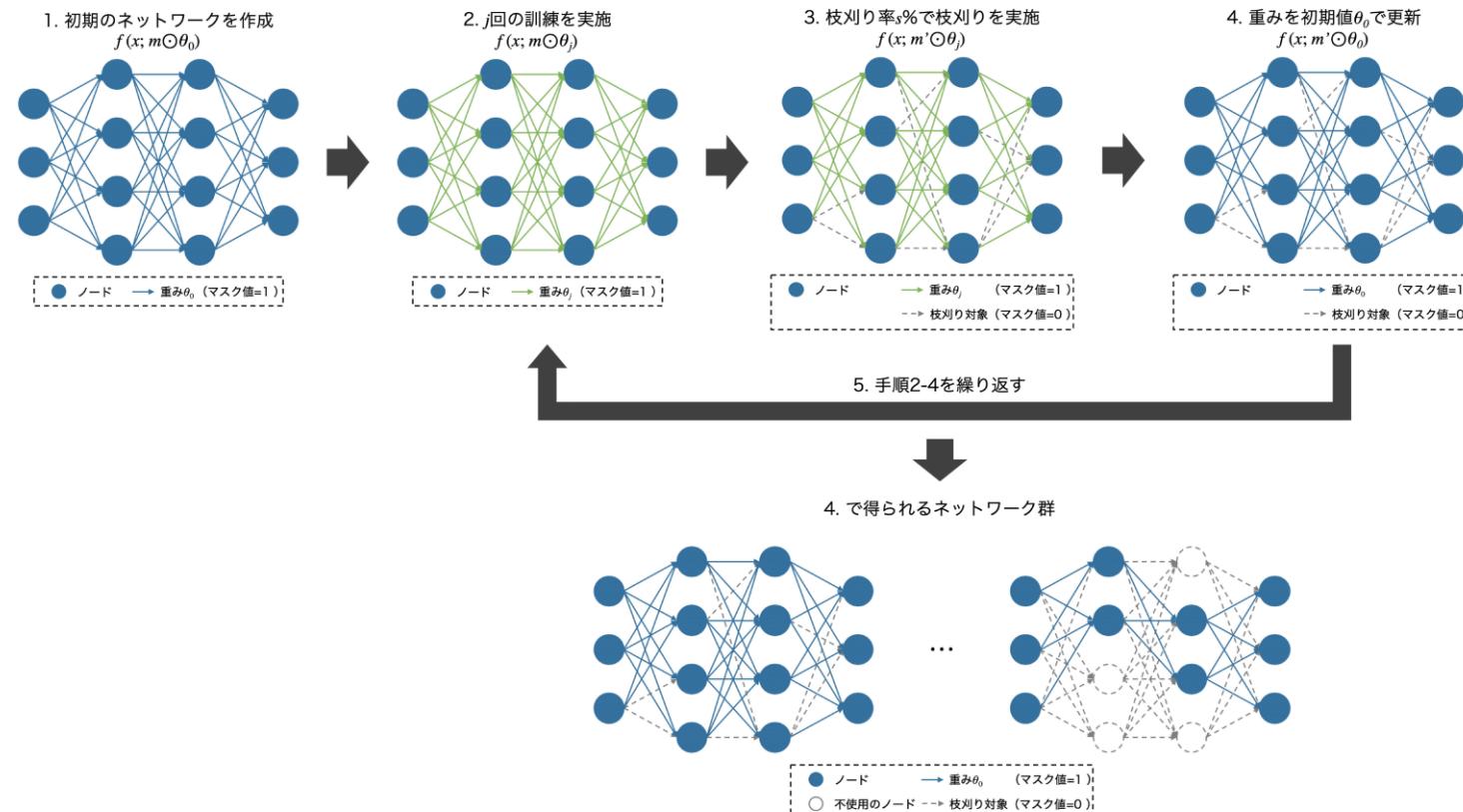
# 目次

- ✓ 宝くじ仮説
  - 宝くじ仮説
  - 当たりくじの流用
  - 強い宝くじ仮説
  - 宝くじ仮説×スケーリング, 損失地形
- ✓ 枝刈り (事例紹介)
  - How to Prune and Distill Llama-3.1 8B to an NVIDIA Llama-3.1-Minitron 4B Model
    - 枝刈りとは (復習) + 枝刈り方法
    - 知識蒸留とは + 知識蒸留方法
    - 結果

～ 宝くじ仮説 ～

# 宝くじ仮説 (LTH : Lottery Ticket Hypothesis)

- ✓ パラメータ過剰なニューラルネットワーク (NN) が汎化する理由は、サブネットワークの候補 (くじ) を大量に持っているから



Pruning後のモデルはサブネットワーク

# ■ 宝くじ仮説 | 背景

- ✓ 何故NNはパラメータ過剰でも汎化性を得るのか?
  - 何故丸暗記を起こさないのか?
- ✓ Knowledge DistillationやPruningから、経験的に仮説は正しい
  - これらは学習後のモデルに用いられる

# ■ 宝くじ仮説 | 提案

## THE LOTTERY TICKET HYPOTHESIS:FINDING SPARSE, TRAINABLE NEURAL NETWORKS, 2019, ICLR Best Paper

- ✓ 大規模なNNには、同じ初期値から学習した場合に同等の性能を達成する**小規模**なサブネットワークが存在する という仮説
- ✓ ↑の検証手順
  - モデルを初期化
  - モデルの初期値を保存, 学習
  - 初期パラメータと学習済みパラメータからマスクを生成
  - マスクの0の割合が一定を超えるまで繰り返し  
  ≒Pruning

# 宝くじ仮説 | 検証結果

- ✓ 枝刈りの過程で性能が向上する場合がある  
⇒ 当たりくじを引いた

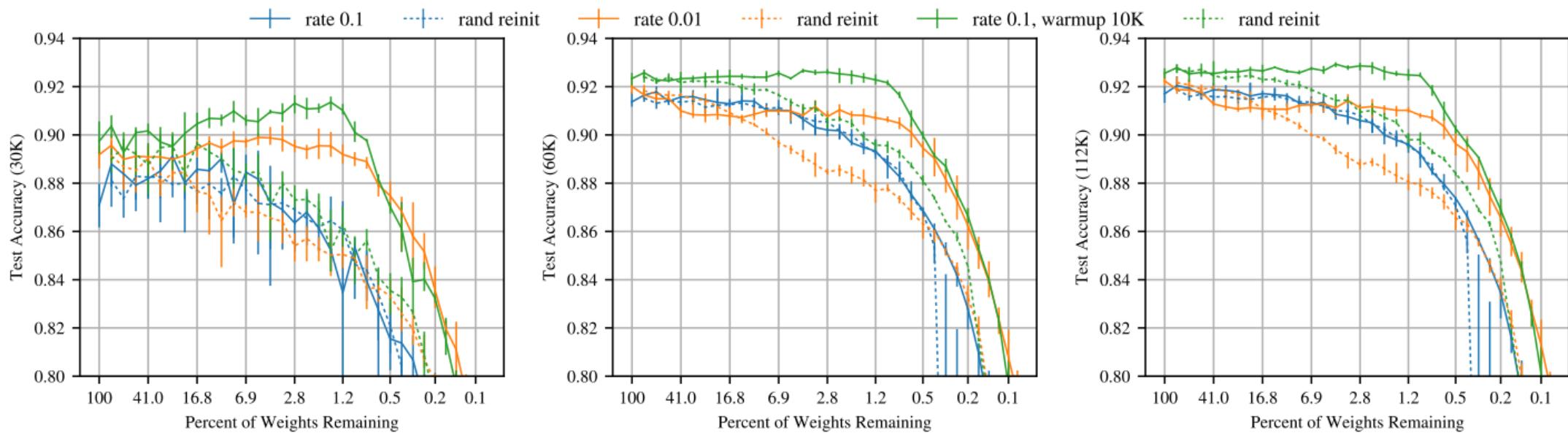


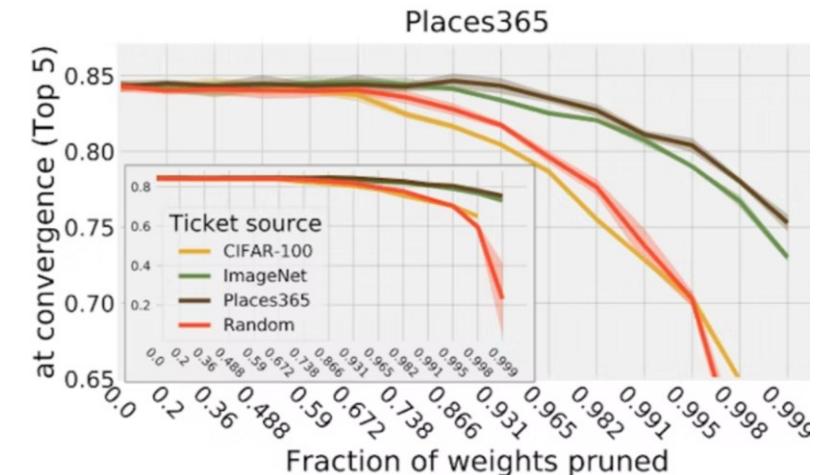
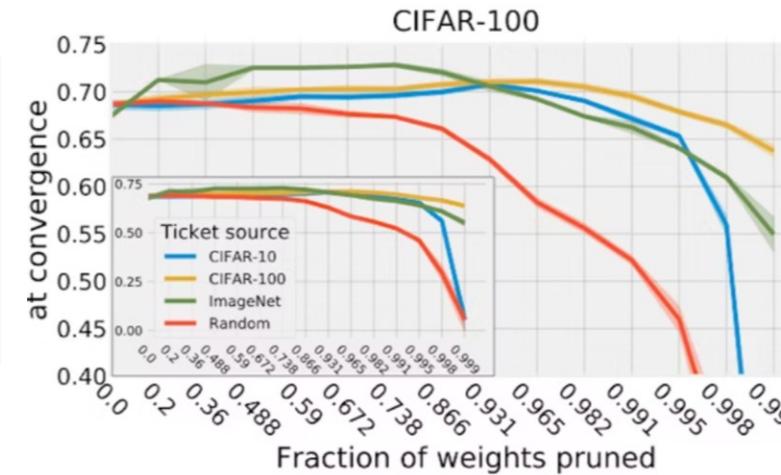
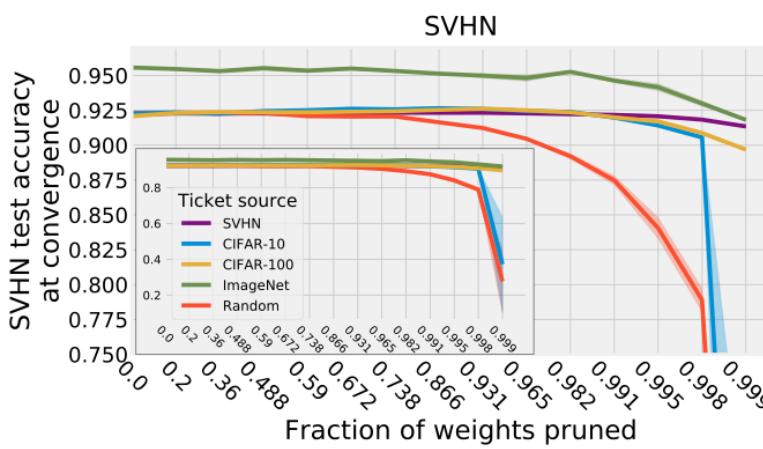
Figure 7: Test accuracy (at 30K, 60K, and 112K iterations) of VGG-19 when iteratively pruned.

同じ実験設定でも性能が変わる ⇒ 初期値依存

# 当たりくじの流用 | データセット転移

One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers, 2019, NeurIPS

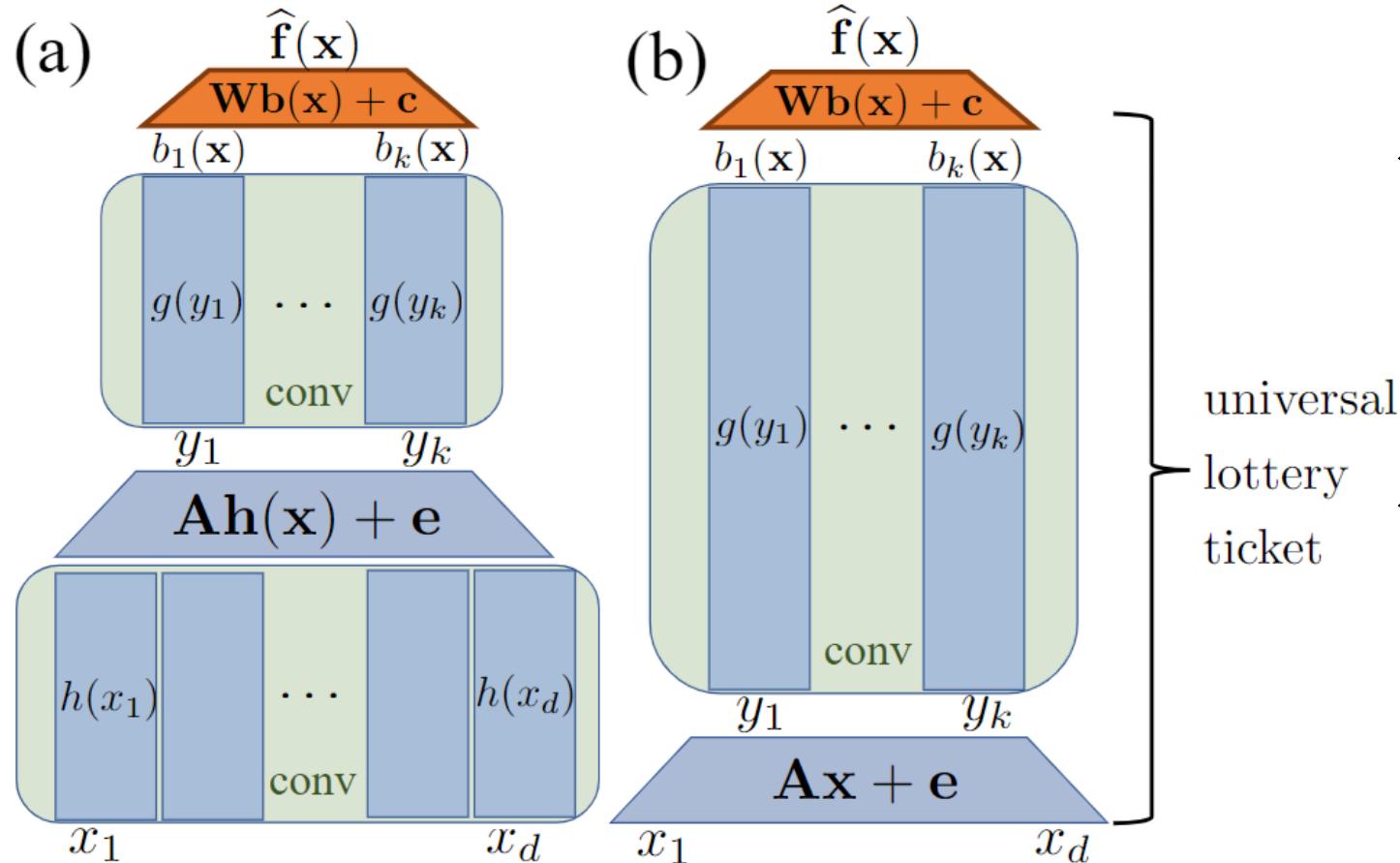
- ✓ 当たりくじのサブネットワークは異なるタスク間で共有できる
  - 別のデータセットで性能が高いNNの初期値（当たりくじ）から下流タスクを学習



データセットが複雑であればあるほど流用のうまみが出る

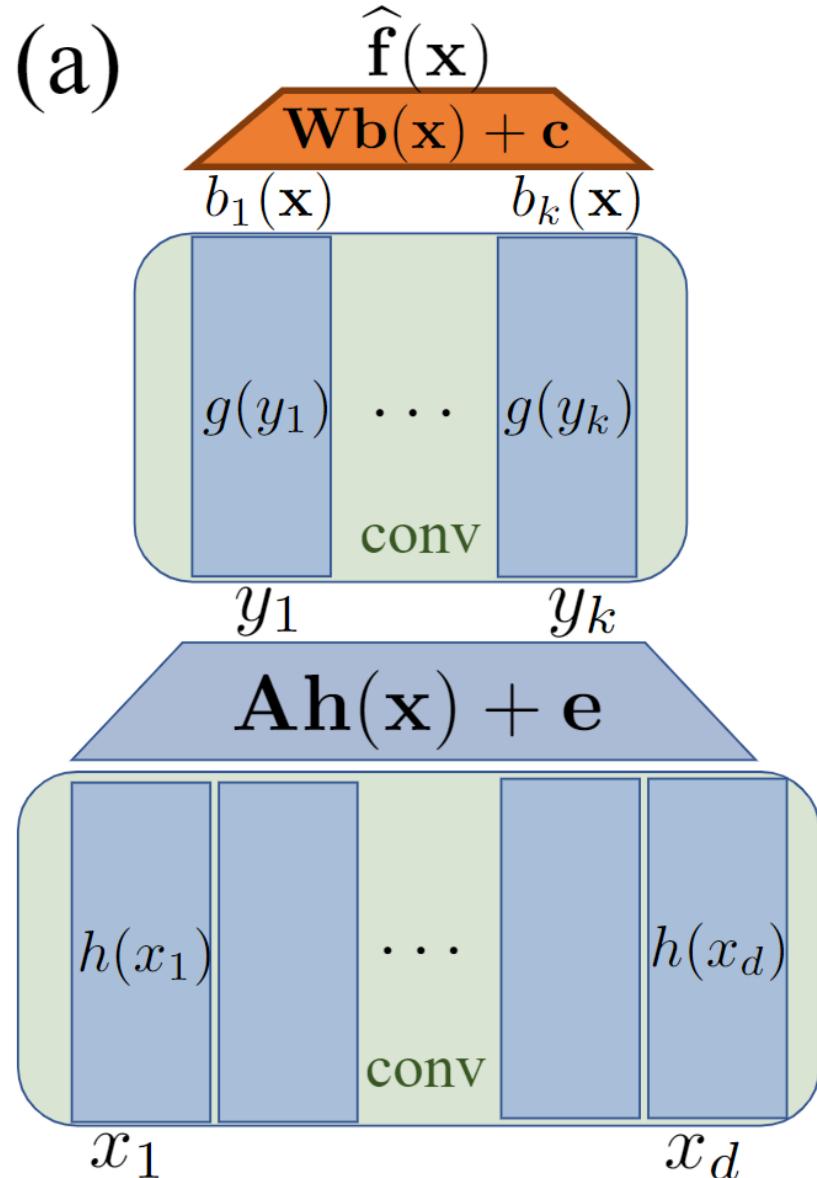
# 当たりくじの流用 | タスク転移 (1/3)

ON THE EXISTENCE OF UNIVERSAL LOTTERY TICKETS, 2022, ICLR



- ✓ 線形変換層の学習のみで複数のタスクに転移可能なサブネットワークの提案
- ✓ バックボーンを2つの方法で再表現することで存在証明

# 当たりくじの流用 | タスク転移 (2/3)

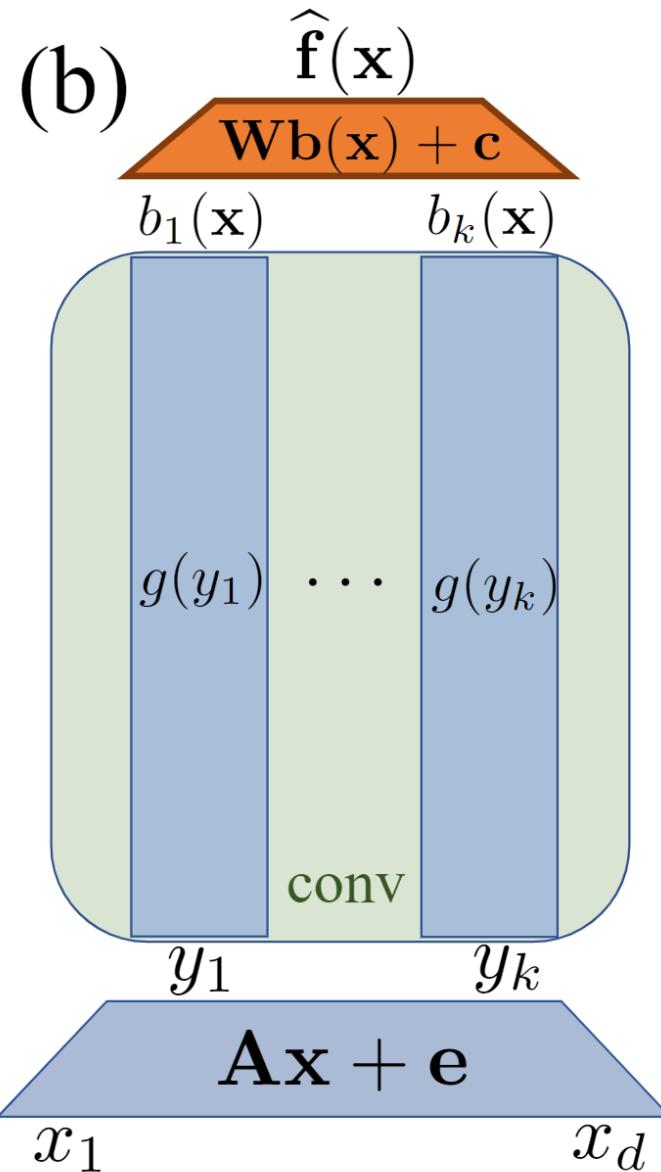


- ✓ ポリノミアルアーキテクチャ
- ✓ CNNの構造に合わせて以下の数式で近似 (定義)

$$b(x) = \exp \left( \sum_{i=1}^d r_i \log(1+x_i) - \log(2) \sum_{i=1}^d r_i \right)$$

- ✓  $\log(1+x_i)$ - $\log(2)$ が置み込み演算
- ✓  $\exp$ が線形結合, 線形変換

# 当たりくじの流用 | タスク転移 (3/3)



- ✓ フーリエ級数アーキテクチャ
- ✓ フーリエ級数の関数表現は以下

$$f(x) = a_0 + \sum_{n \in [N]^d} a_n \sin \left( 2\pi \left( \sum_{i=0}^d n_i x_i + c_n \right) \right) \quad a_n : \text{Fourier series}, \quad c_n : \text{Shift}$$

- ✓ 置み込み層は正弦関数で近似

$$\sin \left( 2\pi \left( \sum_{i=0}^d n_i x_i + c_n \right) \right)$$

- ✓ 線形変換, 線形結合は以下

$$\sum_{i=0}^d n_i x_i + c_n$$

# ■ 強い宝くじ仮説

**What's Hidden in a Randomly Weighted Neural Network?, 2020, CVPR**

## 宝くじ仮説

- ✓ 大規模なNNには、同じ初期値から学習した場合に同等の性能を達成する小規模なサブネットワークが存在する

## 強い宝くじ仮説

- ✓ 大規模な初期値のNNにはサブネットワークが存在し、そのサブネットワークは元と同等の性能を持つ

# 強い宝くじ仮説 | SuperMask

**Deconstructing Lottery Tickets:Zeros, Signs, and the Supermask, 2019, NeurIPS**

- ✓ 初期化済みモデルの重みの一部をマスクし、残りの重みを保持することで**学習を行わずに高精度を達成するSuperMaskを発見**
  - 学習前後で符号が異なる重み、学習後の絶対値が低い重みを0にするマスク
- ✓ 初期化済みモデル（MLP）にSuperMaskを適用したモデルは0-shotでCIFAR10の分類精度65.4%を達成
  - 初期パラメータでコレが達成できることが肝

# 強い宝くじ仮説 | CNN (1/2)

## PROVING THE STRONG LOTTERY TICKET HYPOTHESIS FOR CONVOLUTIONAL NEURAL NETWORKS, 2022, ICLR

- ✓ CNNにおける強い宝くじ仮説を証明 ↓
- ✓ レイヤ*i+1*の入力は前のレイヤの重み乗算 + 活性化で求められる

$$\mathbf{X}_{i+1} = \sigma(\mathbf{K}_i * \mathbf{X}_i)$$

- ✓ ランダムに初期化された畳み込み層 $L_{2i-1}$ と $L_{2i}$ に対応するSuperMask  $\mathbf{S}$ を用いて以下で近似

$$\tilde{\mathbf{X}}_{i+1} = \sigma((\mathbf{L}_{2i} \odot \mathbf{S}_{2i}) * \sigma((\mathbf{L}_{2i-1} \odot \mathbf{S}_{2i-1}) * \mathbf{X}_i))$$

## 強い宝くじ仮説 | CNN (2/2)

- ✓ ランダムに初期化された畳み込み層 $L_{2i-1}$ と $L_{2i}$ に対応するSuperMask  $S$ を用いて以下で近似

$$\tilde{\mathbf{X}}_{i+1} = \sigma((\mathbf{L}_{2i} \odot \mathbf{S}_{2i}) * \sigma((\mathbf{L}_{2i-1} \odot \mathbf{S}_{2i-1}) * \mathbf{X}_i))$$

- ✓ 補題（後述）から、任意の層で以下が成立

$$\sup_{\mathbf{X}} \|\mathbf{K}_i * \mathbf{X} - (\mathbf{L}_{2i} \odot \mathbf{S}_{2i}) * \sigma((\mathbf{L}_{2i-1} \odot \mathbf{S}_{2i-1}) * \mathbf{X})\|_{\max} < \frac{\epsilon}{2\ell}$$

$\epsilon$  : Upper bound  
 $l$  : Number of layers

# 補題 | シングルカーネルの誤差近似

$$\sup_{\mathbf{X} \in [0,1]^{D \times D \times c}} \|\mathbf{K} * \mathbf{X} - g_S(\mathbf{X})\|_{\max} < \epsilon \quad g_S(\mathbf{X}) = \mathbf{V} * \sigma((\mathbf{U} \odot \mathbf{S}) * \mathbf{X})$$

- ✓ 任意の畳み込みカーネル  $\mathbf{K}$  は、ランダムな畳み込みテンソル  $\mathbf{U} \cdot \mathbf{V}$  の適切な Pruning により、ある誤差範囲で近似可能
- ✓ Random Subset Sum Problem で証明可能 (補足資料)

# 補題 | 置み込み層全体の誤差近似

$$\sup_{\mathbf{X} \in [0,1]^{D \times D \times c_0}} \|\mathbf{K} * \mathbf{X} - g_{T,S}(\mathbf{X})\|_{\max} < \epsilon$$

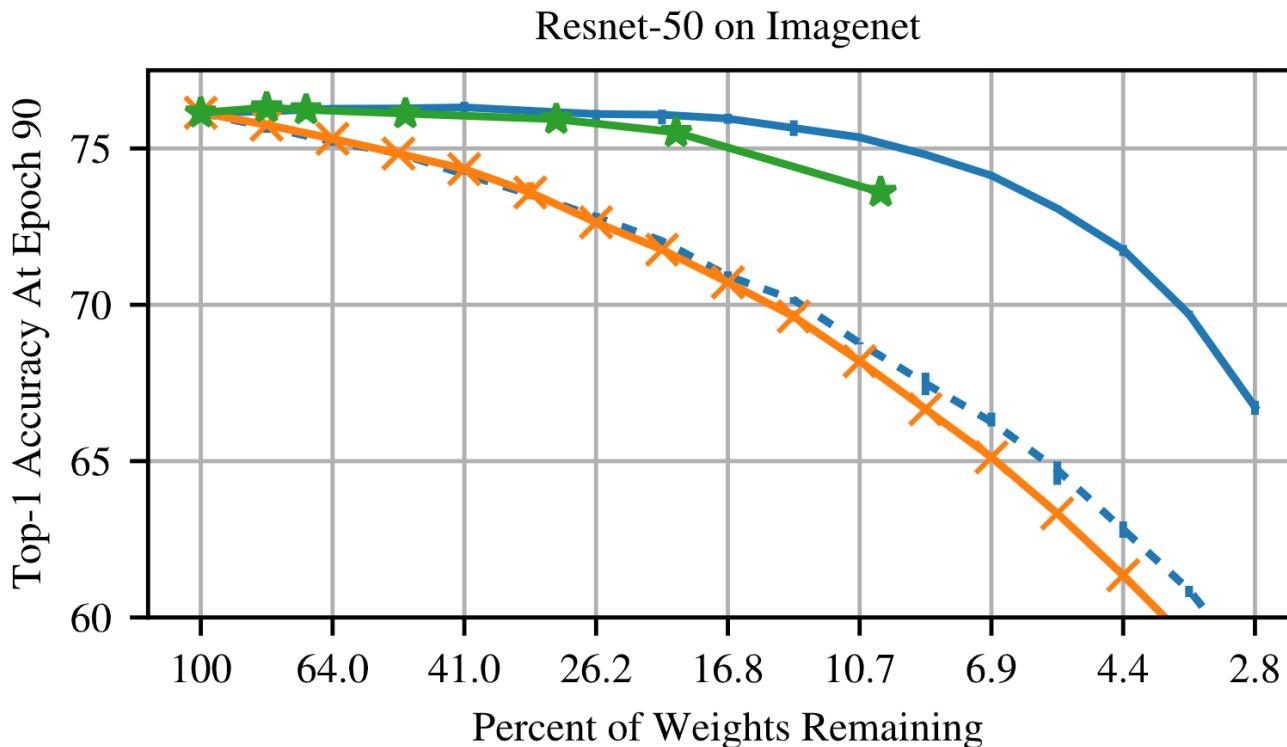
$$g_{T,S}(\mathbf{X}) = (\mathbf{V} \odot \mathbf{T}) * \sigma((\mathbf{U} \odot \mathbf{S}) * \mathbf{X}) \quad \sigma : \text{ReLU}$$

- ✓ 任意の置み込み層は、ランダムに初期化された複数カーネルの適切なPruningにより、ある誤差範囲で近似可能
- ✓ Random Subset Sum Problemで証明可能（補足資料）

# 宝くじ仮説とスケーリング則

## The Lottery Ticket Hypothesis at Scale, 2019, arXiv

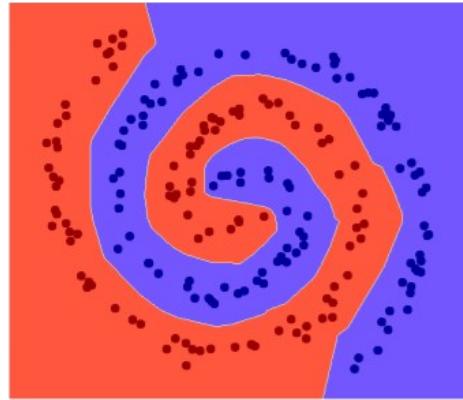
- ✓ 高次のNNはくじが増える分、当たりくじの探索はコストがかかる
- ✓ 最初の数epoch学習した重みを初期値として当たりくじを見つけるLate Resettingを提案



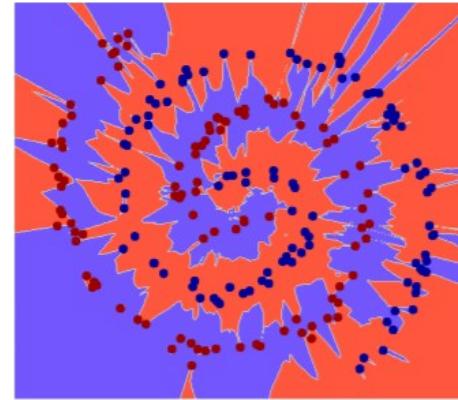
Late Resetting (青・緑実線) は枝刈りしても性能をキープしている  
⇒ 当たりくじを見つけやすい

# ■ 宝くじ仮説と損失地形

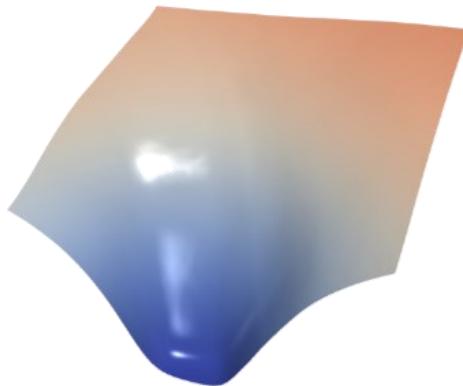
## Understanding Generalization through Visualizations, 2020, arXiv



(a) 100% train, 100% test

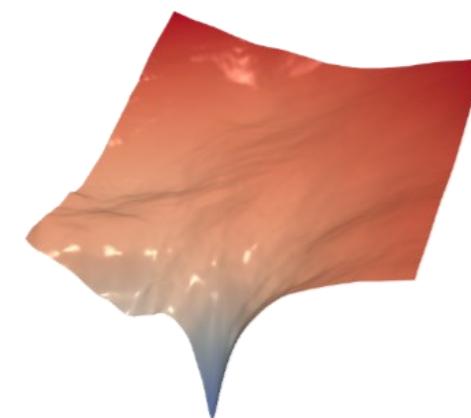


(b) 100% train, 7% test



(c) Minimizer of network in (a) above

汎化性を持った学習



(d) Minimizer of network in (b) above

過学習

✓ 学習済みNNがテストデータに対し汎化性を持つ要因を調査

✓ 損失周辺の形状を可視化し定量化することで汎化性を評価

# 損失地形の定量化 (1/2)

- ✓ basinの体積から極小値を定量化

$$V = \omega_n \mathbb{E}_\phi [r^n(\phi)] \quad \omega_n = \frac{\pi^{n/2}}{\Gamma(1 + \frac{n}{2})}$$

$$r(\phi) = \min \{r \mid L(\theta^* + r\phi) > L(\theta^*) + \epsilon\}$$

- ✓ basin : 損失関数の極小値周辺領域

$$\mathcal{B}(\theta^*) = \{\theta \mid L(\theta) \leq L(\theta^*) + \epsilon\}$$

$r$  : Radius  
 $n$  : Dimension  
 $\phi$  : Direction  
 $B$  : Basin  
 $\theta^*$  : Model parameter of minima  
 $\theta$  : Model parameter  
 $\epsilon$  : Threshold

## ■ 損失地形の定量化 (2/2)

✓ 完璧な体積を求めることは**不可能**

–  $n$ はモデルのパラメータ次元 = 数千万～数億以上

$$V = \omega_n \mathbb{E}_\phi [r^n(\phi)] \quad r(\phi) = \min \{r \mid L(\theta^* + r\phi) > L(\theta^*) + \epsilon\}$$

✓ 実際の手順

–  $\phi$ をモンテカルロ法でサンプリング

–  $r$ を計算  $\Rightarrow r^n$ を計算  $\Rightarrow$  平均を計算

✓ 体積が大きい  $\Rightarrow$  地形が平坦  $\Rightarrow$  良い解にたどり着いている

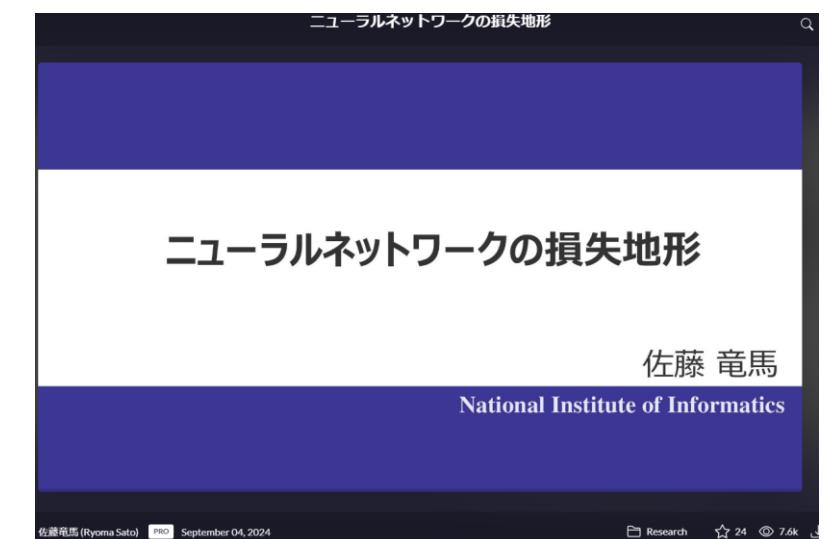
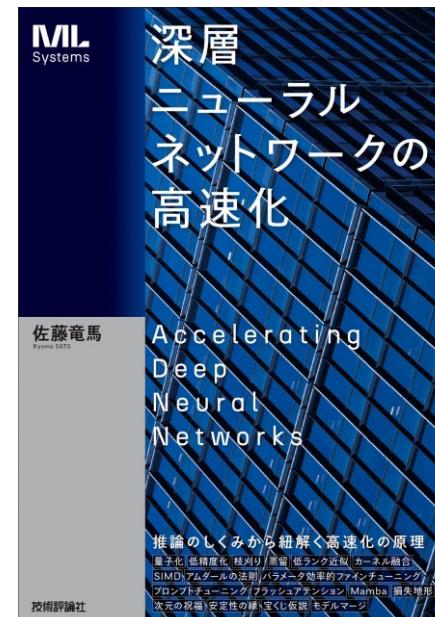
✓ 次元が大きい  $\Rightarrow$  basinの向きが増える  $\Rightarrow$  地形が広がりやすい

$\Rightarrow$  最初のNNの次元は大きくするべき = 宝くじ仮説と同様の結論

# 小まとめ

- ✓ NNには同じ性能を達成可能なサブネットワークが存在する
- ✓ 宝くじ仮説に従うと、NNは初期値問題から逃げられない
  - 少しでも緩和するために、次元を増やす・探索を行う・学習済みモデルを使う
- ✓ 損失地形の観点からも初期NNの次元を増やすことは有効
  - 地形を均すアプローチ (SAM) なども
  - 後から小さくする ⇒ Pruning

参考 ⇒



<https://speakerdeck.com/joisino/landscape>

～枝刈り（事例紹介）～

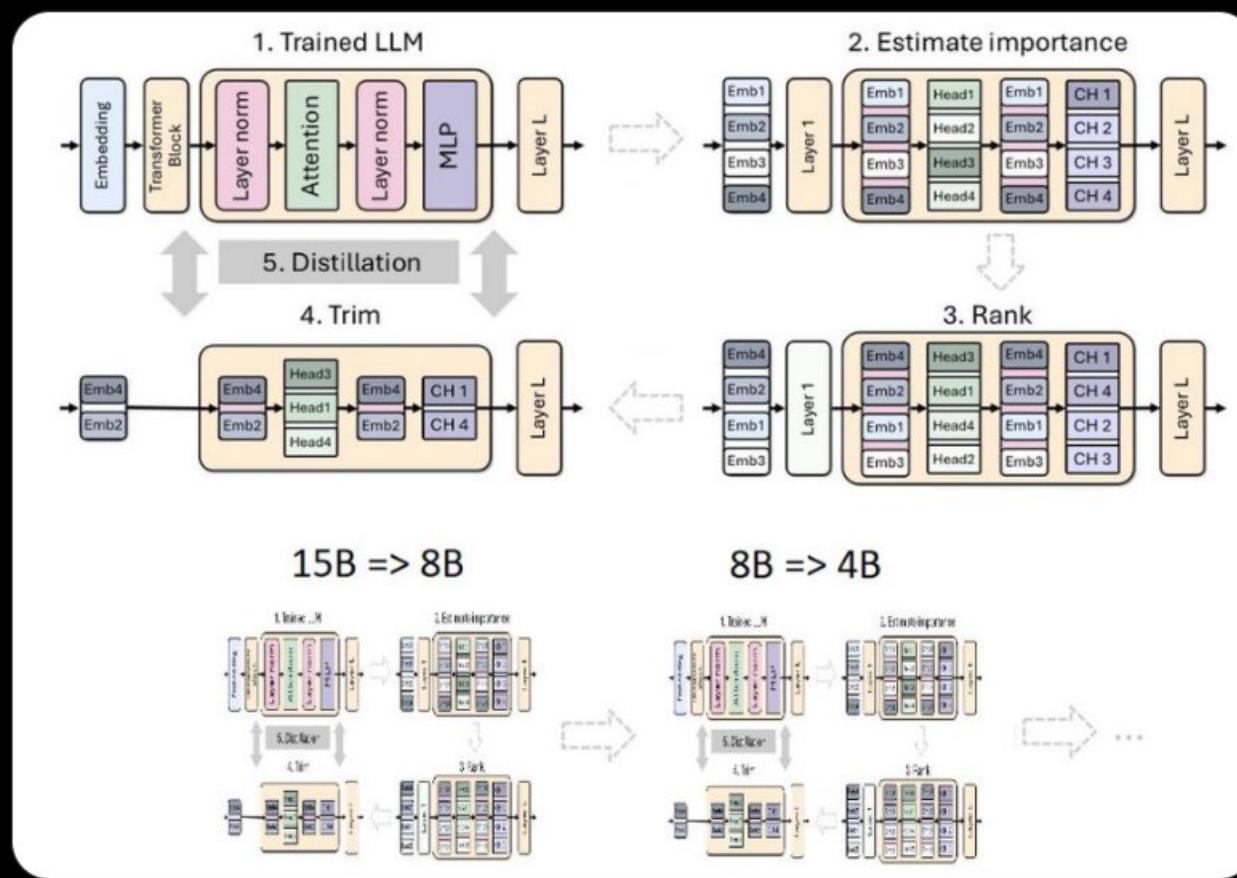
# How to Prune and Distill Llama-3.1 8B to an NVIDIA Llama-3.1-Minitron 4B Model



AI at Meta ✨ 💡 @AIatMeta · 8月16日

Using structured weight pruning and knowledge distillation, the @NVIDIAAI research team refined Llama 3.1 8B into a new Llama-3.1-Minitron 4B.

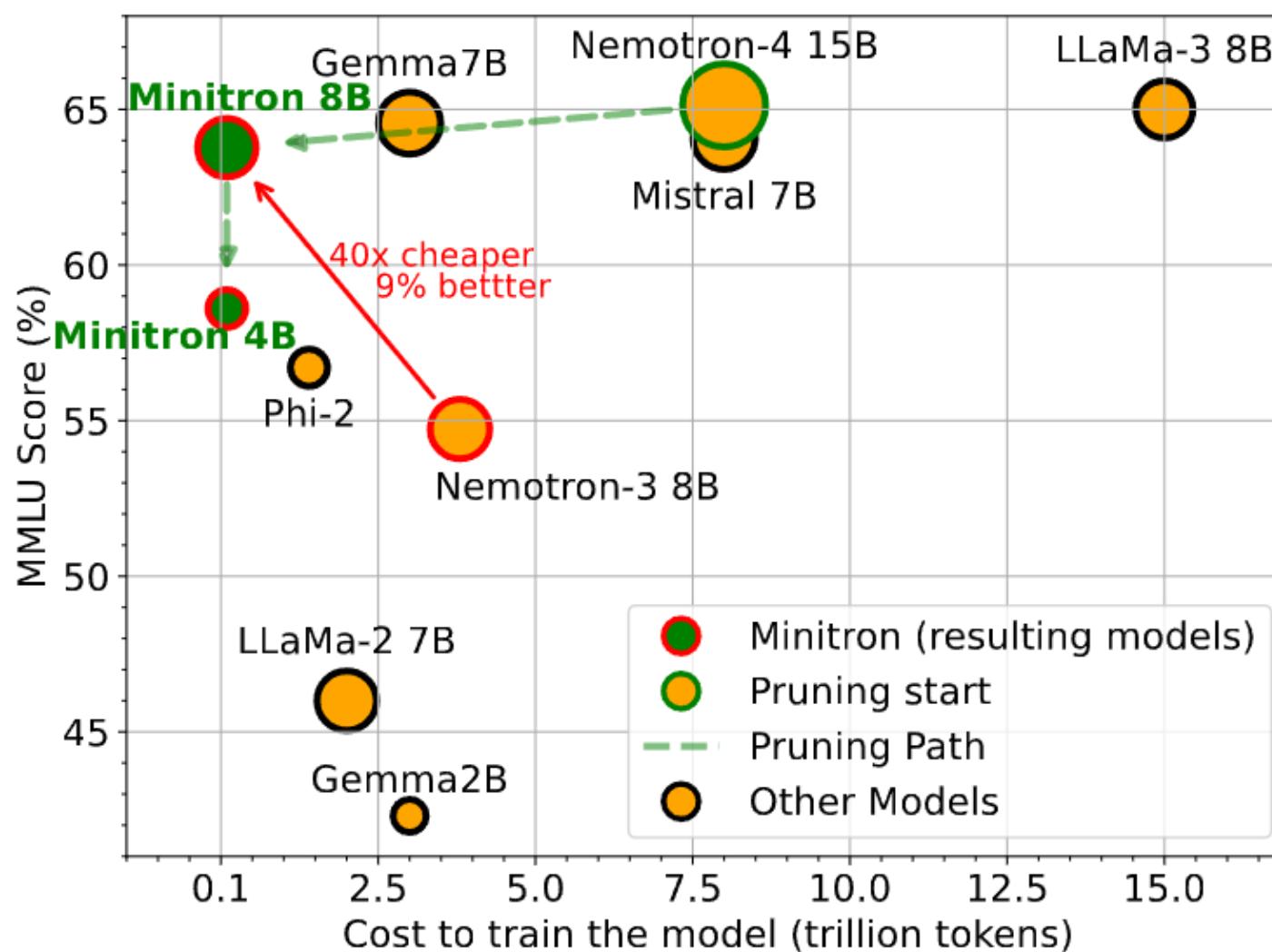
They're releasing the new models on @huggingface and shared a deep dive on how they did it ➡ [go.fb.me/b2h2c8](https://go.fb.me/b2h2c8)



✓ NVIDIA, Pruningと  
Distillationにより  
LLMを小規模化

- Pruning : 枝切り
- Distillation : 知識蒸留
- LLaMA 3.1 15B → 8B
- LLaMA 3.1 8B → 4B

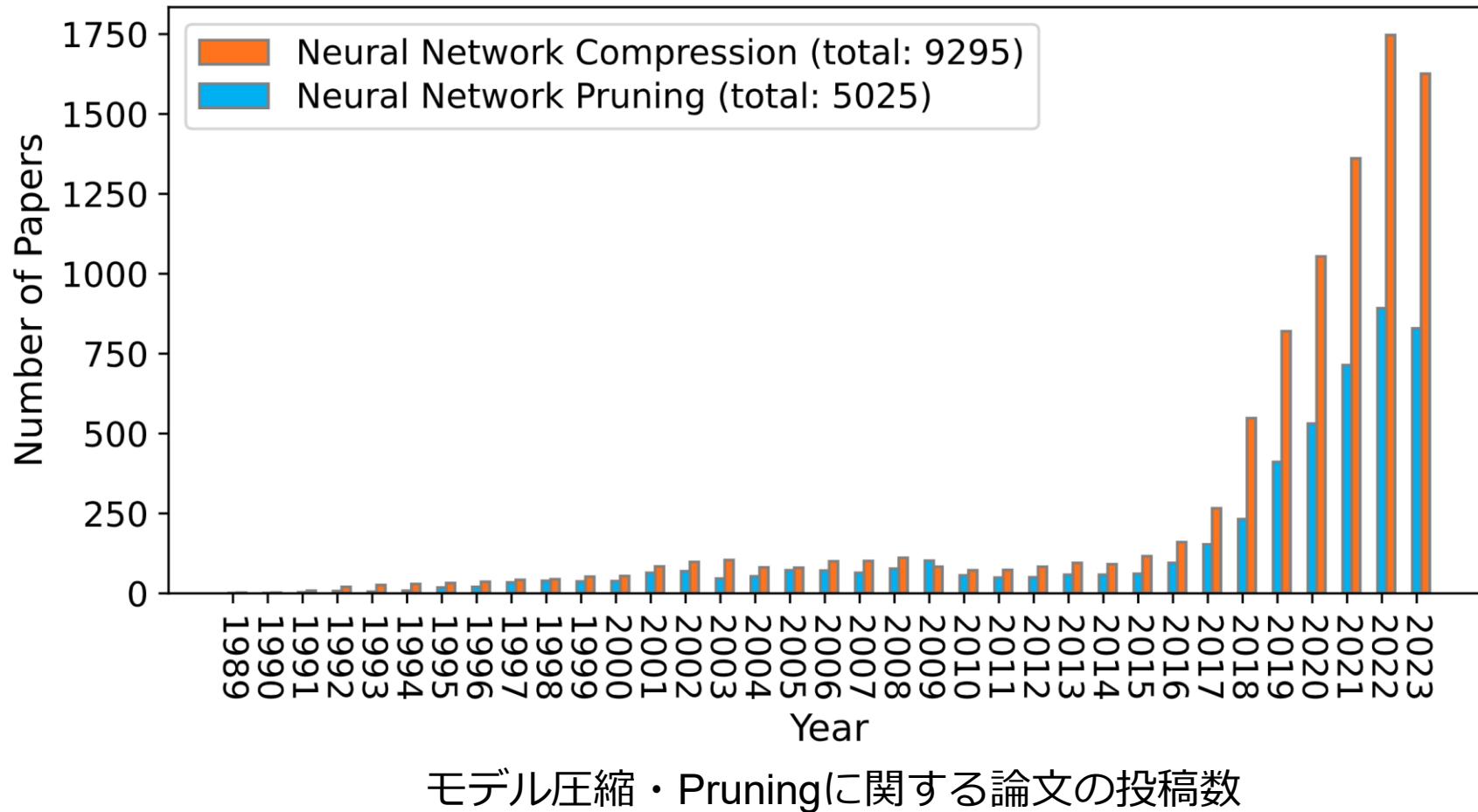
# Compact Language Models via Pruning and Knowledge Distillation, 2024, arXiv



- ✓ PruningとDistillationを用いて LLMの小規模化を達成
  - Nemortron 15B → Minitron 8B
  - Nemotron 8B → Minitron 4B
  - Minitron 8B → Minitron 4B
  - Nemotron: NVIDIA製LLM

# Pruningとは (1/2)

- ✓ Pruning (枝刈り) : モデルから不要なノードを削除することでモデルの小規模化・高速化を行う技術



# Pruningとは (2/2)

元の  
パラメータ行列

$$\begin{pmatrix} 1.23 & 0.24 & -0.21 \\ 5.65 & 7.81 & 0.12 \\ 0.03 & -3.43 & 5.52 \end{pmatrix}$$



疎行列

$$\begin{pmatrix} 1.23 & 0 & 0 \\ 5.65 & 7.81 & 0 \\ 0 & -3.43 & 5.52 \end{pmatrix}$$

**非構造枝刈り**  
0の要素の計算を省略可能

$$\begin{pmatrix} 1.23 & 0.24 \\ 5.65 & 7.81 \\ 0.03 & -3.43 \end{pmatrix}$$

**構造枝刈り**  
行や列の単位で枝刈り

**枝刈りにより計算量を削減  
(1~3倍程度の高速化)**

# Pruning方法 | 事前定義

MLP

$$\text{MLP}(\mathbf{X}) = \delta(\mathbf{X} \cdot \mathbf{W}_1^T) \cdot \mathbf{W}_2$$

Attention

$$\begin{aligned}\text{MHA}(\mathbf{X}) &= \text{Concat}(\text{head}_1, \dots, \text{head}_L) \cdot \mathbf{W}^O, \\ \text{head}_i &= \text{Attn}(\mathbf{X}\mathbf{W}^{Q,i}, \mathbf{X}\mathbf{W}^{K,i}, \mathbf{X}\mathbf{W}^{V,i})\end{aligned}$$

Layer Normalization

$$\text{LN}(\mathbf{X}) = \frac{\mathbf{X} - \mu}{\sqrt{\sigma^2 + \epsilon}} \circ \gamma + \beta$$

$X$  : Input

$W$  : Model Weight

$\delta$  : Activate Function

$\mu$  : Mean of Embedding

$\sigma^2$  : Variance of Embedding

$\epsilon$  : Hyperparameter(small)

$\gamma, \beta$  : Learnable parameter

# Pruning方法 | 幅方向の探索

Head, Neuron, Embedding channelに対する重要度算出は以下

$$F_{\text{head}}^{(i)} = \sum_{\mathbf{B}, \mathbf{S}} \|\text{Attn}(\mathbf{X}\mathbf{W}^{Q,i}, \mathbf{X}\mathbf{W}^{K,i}, \mathbf{X}\mathbf{W}^{V,i})\|_2$$
$$F_{\text{neuron}}^{(i)} = \sum_{\mathbf{B}, \mathbf{S}} \mathbf{X}(\mathbf{W}_1^i)^T, \quad F_{\text{emb}}^{(i)} = \sum_{\mathbf{B}, \mathbf{S}} \text{LN}(\mathbf{X})_i$$

Batch	Sequence	8T LM Loss	WikiText2 LM Loss
L2	L2	8.73	8.37
<b>L2</b>	<b>mean</b>	<b>7.18</b>	<b>7.23</b>
L2	var	8.18	8.61
mean	L2	8.41	7.84
<b>mean</b>	<b>mean</b>	<b>7.21</b>	<b>6.89</b>
mean	var	7.94	8.29
var	L2	9.01	9.30
var	mean	8.34	8.72
var	var	10.55	11.14

集計方法を変更したときの性能比較

集計方法は  
Mean, Variance, L2の  
Grid Search

# Pruning方法 | 深さ方向の探索

## PerplexityとBlock Importanceで層の重要度算出

- Perplexity: 特定の層を除外した際のPerplexityスコアの変動値

$$\text{ppl} = \exp \left( -\frac{1}{N} \sum_n \sum_k t_{n,k} \log p_{\text{model}}(y_{n,k}) \right)$$

$n$  : index of output token  
 $k$  : index of token list

- Block Importance: ある層について入力前後の特徴量間のCosine Distance

$$\text{BI}_i = 1 - \mathbb{E}_{\mathbf{X},t} \frac{\mathbf{X}_{i,t}^T \mathbf{X}_{i+1,t}}{\|\mathbf{X}_{i,t}\|_2 \|\mathbf{X}_{i+1,t}\|_2}$$

# Pruning方法 | イテレーションごとのPruning

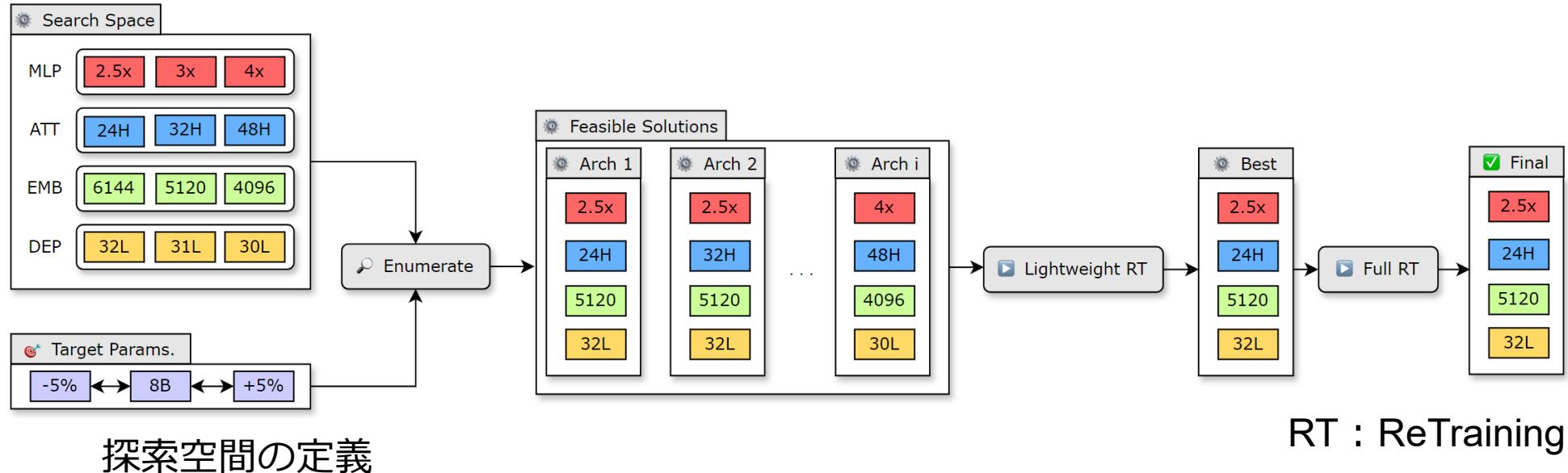
- ✓ Tイテレーション毎に重要度算出・Pruningを実行

Iterations	Initial (Zero-Shot) Validation Loss	Final Validation Loss
T=1	5.43	1.92
T=2	5.55	1.92
T=4	<b>5.24</b>	1.92

Tを変化させたときの性能比較

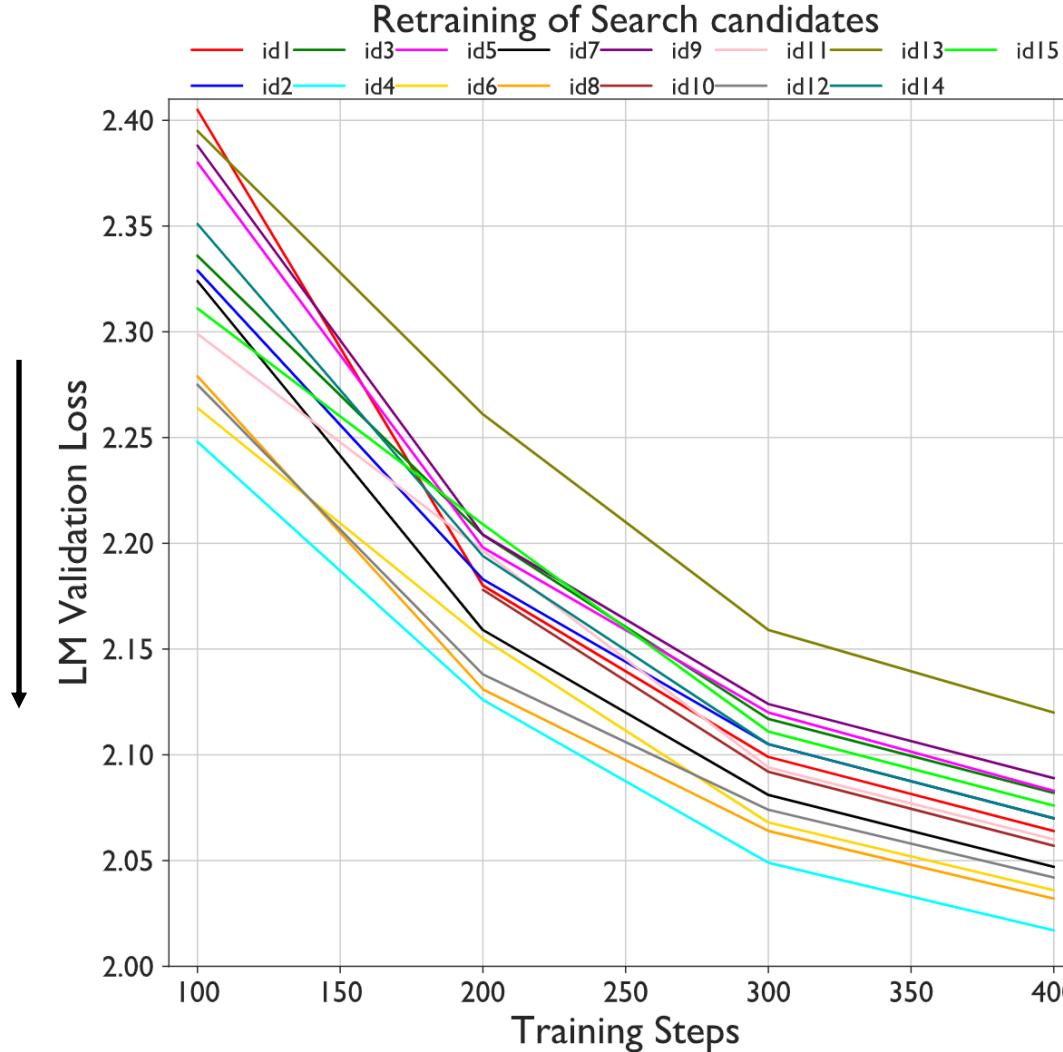
# Pruning方法 | Neural Architecture Search (1/2)

## 軽量のNeural Architecture Search (NAS)



- ✓ Pruning後のモデルに対してNASを適用しモデル構造を再構築
- NAS : モデル構造の探索アルゴリズム

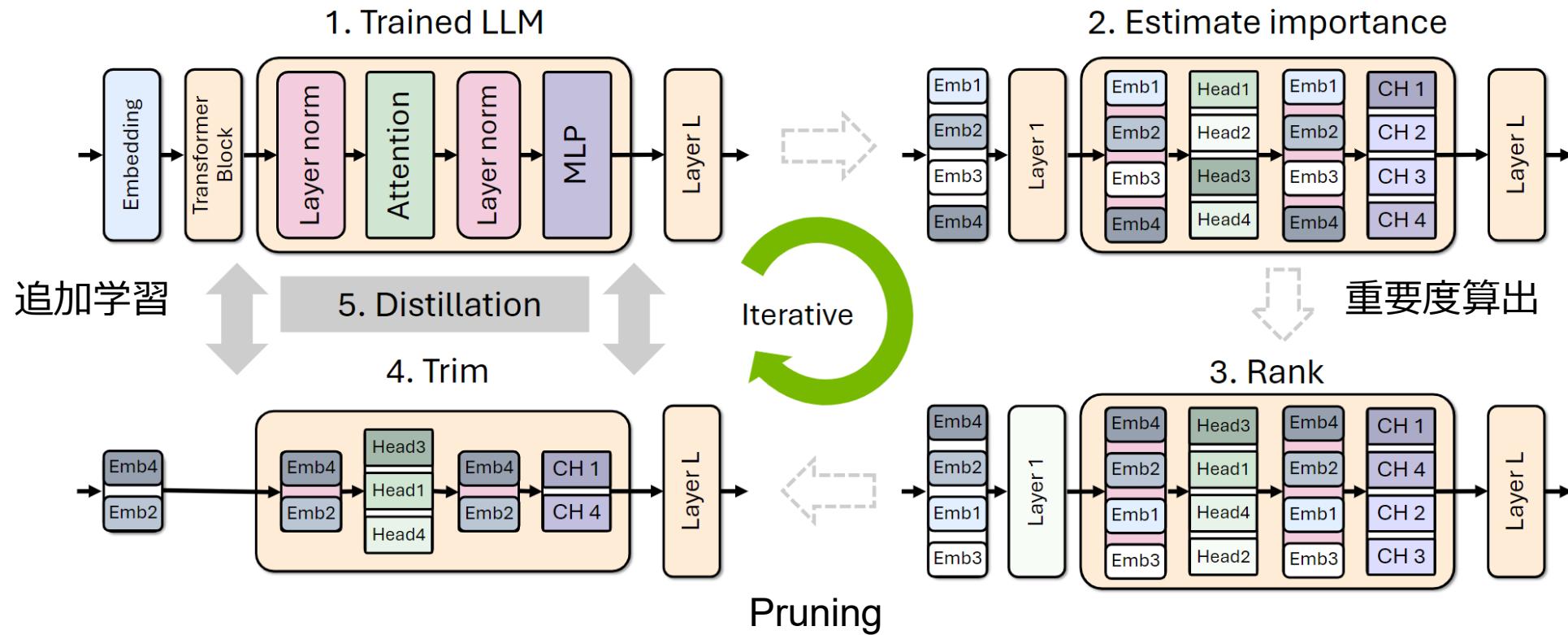
# Pruning方法 | Neural Architecture Search (2/2)



候補モデル探索の性能比較

候補モデルによって性能がばらつく ⇒ 探索の有効性を確認

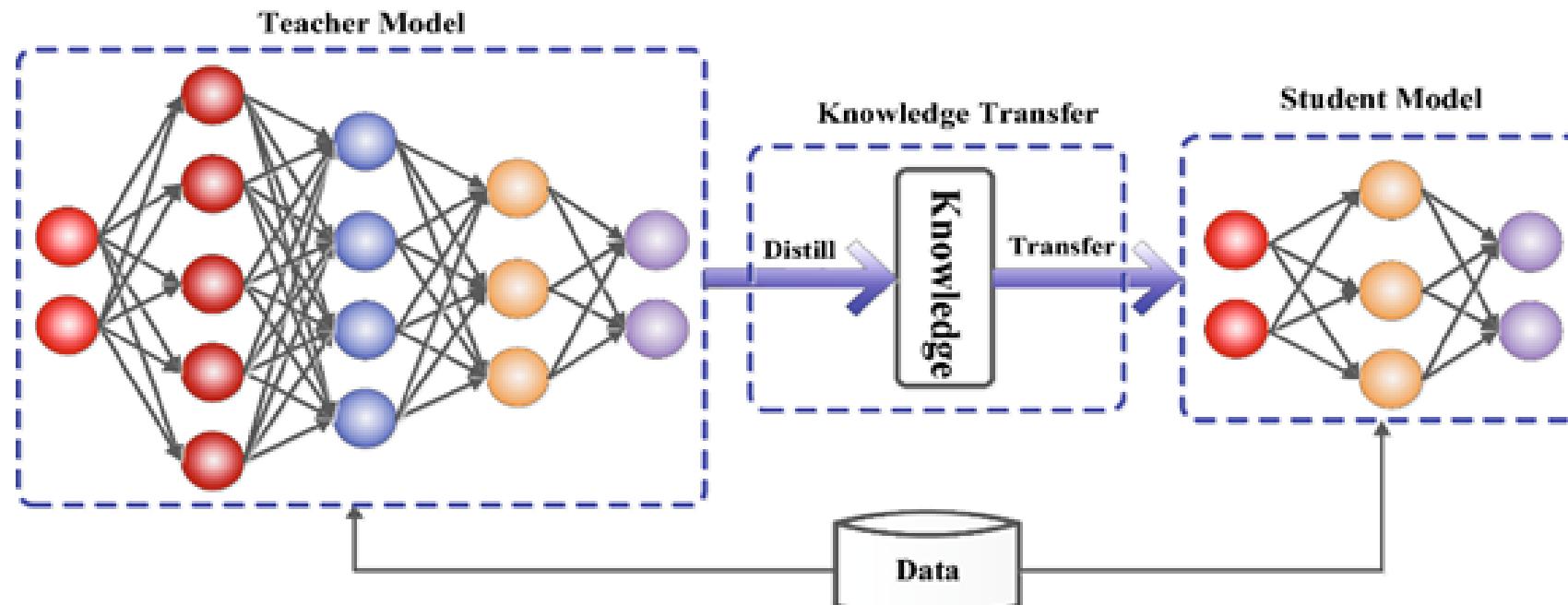
# Pruning方法 | Pruningされたモデルの獲得



PruningされたAttention Headからは残差接続

# Distillationとは (1/2)

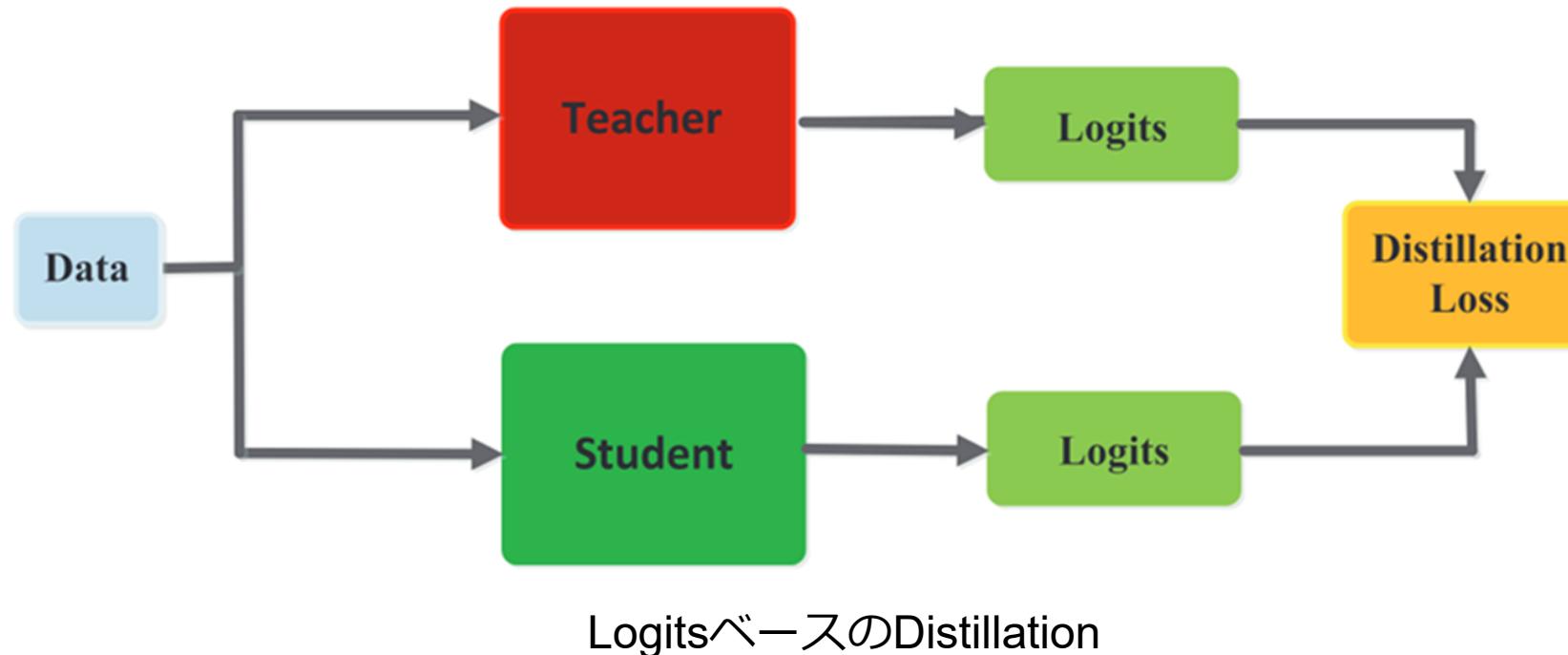
- ✓ **Distillation (蒸留)** : 巨大な学習済みモデル（教師）を小規模なモデル（生徒）で近似するように学習する技術



Distillation イメージ図

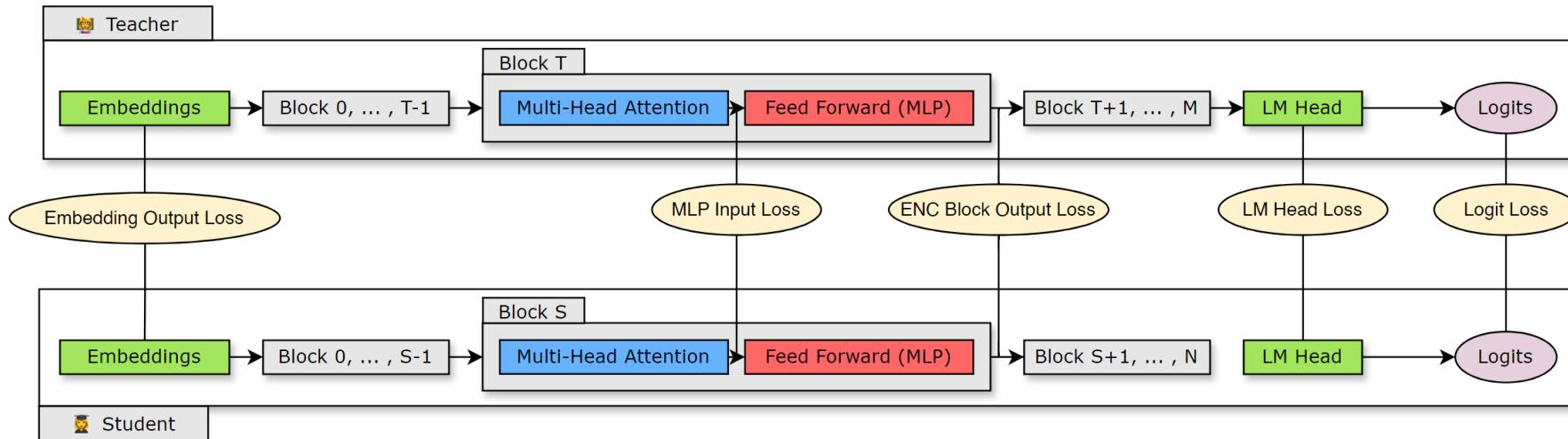
# Distillationとは (2/2)

- ✓ **Distillation (蒸留)** : 巨大な学習済みモデル（教師）を小規模なモデル（生徒）で近似するように学習する技術



# Distillation方法 (1/4)

Pruning前を教師，Pruning後を生徒としたKnowledge Distillation



- ✓ 中間表現，Logitのそれぞれに対して最適化を行うことで，教師モデルを再現する生徒モデルを獲得

## Distillation方法 (2/4)

Pruning前を教師， Pruning後を生徒としたKnowledge Distillation

✓ LogitベースのKnowledge Distillation

$$L_{\text{logits}} = \frac{1}{l} \sum_{k=1}^l \text{Loss}(p_t^k(x, \tau), p_s^k(x, \tau))$$

$$p(x_i, \tau) = \frac{\exp\left(\frac{x_i}{\tau}\right)}{\sum_{j=1}^{|V|} \exp\left(\frac{x_j}{\tau}\right)}$$

$l$  : Sequence length

$\tau$  : Temperature

$|V|$  : Vocabulary size

Loss : KLD, MSE, Cosine Similarity, ReverseKLD

※ 同時に教師あり学習も実行

# Distillation方法 (3/4)

## LogitベースのKnowledge Distillation

- ✓ Nemotron-3 8Bで最適なlogitsのLossを探索

Loss	LM loss	WikiText PPL
$L_{CLM} + L_{logits}(\text{MSE})$	2.144	9.007
$L_{CLM} + L_{logits}(\text{RKLD})$	2.140	9.008
$L_{CLM} + L_{logits}(\text{Cosine})$	2.134	8.965
$L_{CLM} + L_{logits}(\text{KLD})$	<b>2.117</b>	<b>8.791</b>
$L_{logits}(\text{KLD})$	<b>2.107</b>	<b>8.720</b>

Lossを変更したときの性能比較

# Distillation方法 (3/4)

## Pruning前を教師，Pruning後を生徒としたKnowledge Distillation

- ✓ Transformerベースモデルの中間表現のKnowledge Distillation

$$L_{is} = \frac{1}{l} \sum_{k \in H} \sum_{i=1}^l \text{Loss}_k(h_t^{ki}, h_s^{ki})$$

$h$  : Hidden state for  $i^{th}$  token

$$L = L_{\text{CELstudent}} + L_{\text{logits}} + \alpha \times L_{\text{is}}$$

- $\alpha$ はlogitsとisの比から計算する場合も
- $L_{\text{is}}$ はembedding, attention, mlp, outputから探索

# Distillation方法 (4/4)

## 中間表現のKnowledge Distillation

- ✓ Nemotron-4 15Bで最適な中間表現のLossを探索

Loss components	LM loss
$L_{logits}$	2.155
$L_{logits} + L_o(29:13)$	<b>2.145</b>
$L_{logits} + L_o(15:15) + L_{emb}$	2.240
$L_{logits} + L_o(23:15) + L_{emb}$	2.205
$L_{logits} + L_o(29:15) + L_{emb}$	2.203
$L_{logits} + L_o(30:15) + L_{emb}$	2.188
$L_{logits} + L_o(31:15) + L_{emb}$	2.180
$L_{logits} + L_o(28:12) + L_{emb}$	2.141
$L_{logits} + L_o(29:13) + L_{emb}$	<b>2.141</b>
$L_{logits} + L_o(29:14) + L_{emb}$	2.152
$L_{logits} + L_o(30:14) + L_{emb}$	2.150
$L_{logits} + L_o(29:13) + L_{emb} + L_i(29:13)$	2.141 (教師の層 : 生徒の層)

Lossを変更したときの性能比較

# 実験

## モデル

✓ Nemotron-4 15BをPruningしてMINITRONを構築

## データセット

✓ 事前学習： Nemotron-4からcurationした8T トークンのテキスト

✓ ReTraining： ↑から1.8B トークン

✓ 重要度計算： ↑から1,024サンプル

1. To train a family of LLMs, train the largest one and prune+distill iteratively to smaller LLMs.
2. Use (batch=L2, seq=mean) importance estimation for width axes and PPL/BI for depth.
3. Use single-shot importance estimation; iterative provides no benefit.
4. Prefer width pruning over depth for the model scales we consider ( $\leq 15B$ ).
5. Retrain exclusively with distillation loss using KLD instead of conventional training.
6. Use (logit+intermediate state+embedding) distillation when depth is reduced significantly.
7. Use logit-only distillation when depth isn't reduced significantly.
8. Prune a model closest to the target size.
9. Perform lightweight retraining to stabilize the rankings of searched pruned candidates.
10. If the largest model is trained using a multi-phase training strategy, it is best to prune and retrain the model obtained from the final stage of training.

# 結果 (1/4)

Model	Tokens	Hellaswag	MMLU
4B-Random-Init	150B*	46.22	24.36
4B-Random-Init	400B	48.23	26.24
4B-Pruned (prune Nemotron-4 15B)	150B*	50.85	24.57
<b>4B-Pruned-Distill (prune Nemotron-4 15B)</b>	<b>100B*</b>	<b>51.04</b>	<b>37.81</b>
<b>4B-Pruned-Distill (prune MINITRON 8B)</b>	<b>100B*</b>	<b>52.04</b>	<b>42.45</b>

Pruning方法の比較結果

- ✓ Distillationによる性能向上・訓練データの削減を確認

# 結果 (2/4)

		Models							
	Benchmark	Metric	Llama-3	Llama-2	Mistral	Gemma	Nemotron-4	Nemotron-3	MINITRON
	# Parameters		8B	6.7B	7.3B	8.5B	15.6B	8.5B	8.3B
	# Non-Emb. Params		5.9B	6.4B	7B	7.7B	12.5B	6.4B	6.2B
	# Training Tokens		>15T	2T	8T	6T	8T	3.8T	<b>94B</b>
Knowledge, Logic	winogrande (5)	acc	78	74	78.5	78	83.6	75.9	<b>79.0</b>
	arc_challenge (25)	acc_norm	58	53	60.3	<b>61</b>	58.8	52.8	52.6
	MMLU(5)	acc	<b>65</b>	46	64.1	64	66.6	54.7	63.8
	hellaswag(10)	acc_norm	82	79	<b>83.2</b>	82	84.6	78.5	80.7
	gsm8k(5)	acc	50	14	37	50	48.5	24.0	<b>51.3</b>
	truthfulqa(0)	mc2	44	39	42.6	<b>45</b>	40.7	36.5	42.6
	XLSum en (20)(3)	rougeL	31	31	4.80	17	32	30.9	<b>31.2</b>
Coding	MBPP(0)	pass@1	<b>42</b>	20	38.8	39	38	27.04	35.2
	humaneval (n=20)(0)	pass@1	28	12	28.7	<b>32</b>	35.4	20.7	31.6

MINITRON 8B 他のLLMとの比較

- ✓ 非常に少ないトークンで学習可能

# 結果 (3/4)

			Models					
	Benchmark	Metric	Phi-2	Gemma	Gemma2*	Qwen2*	MiniCPM*	MINITRON
# Parameters			2.7B	2.5B	2.6B	1.5B	2.7B	4.2B
# Non-Emb. Params			2.5B	2B	2B	1.3B	2.4B	2.6B
# Training Tokens			1.4T	3T	2T	7T	1.1T	<b>94B</b>
Knowledge, Logic	winogrande (5)	acc	<b>74</b>	67	70.9	66.2	-	<b>74</b>
	arc_challenge (25)	acc_norm	<b>61</b>	48	55.4	43.9	-	50.9
	MMLU(5)	acc	57.5	42	51.3	56.5	53.5	<b>58.6</b>
	hellaswag(10)	acc_norm	<b>75.2</b>	72	73	66.6	68.3	75
	gsm8k(5)	acc	55	18	23.9	<b>58.5</b>	53.8	24.1
	truthfulqa(0)	mc2	44	33	-	<b>45.9</b>	-	42.9
	XLSum en (20)(3)	rougeL	1	11	-	-	-	<b>29.5</b>
Coding	MBPP(0)	pass@1	<b>47</b>	29	29.6	37.4	-	28.2
	humaneval (n=20)(0)	pass@1	<b>50</b>	24	17.7	31.1	-	23.3

MINITRON 4B 他の類似したサイズのLLMとの比較

- ✓ 非常に少ないトークンで学習可能

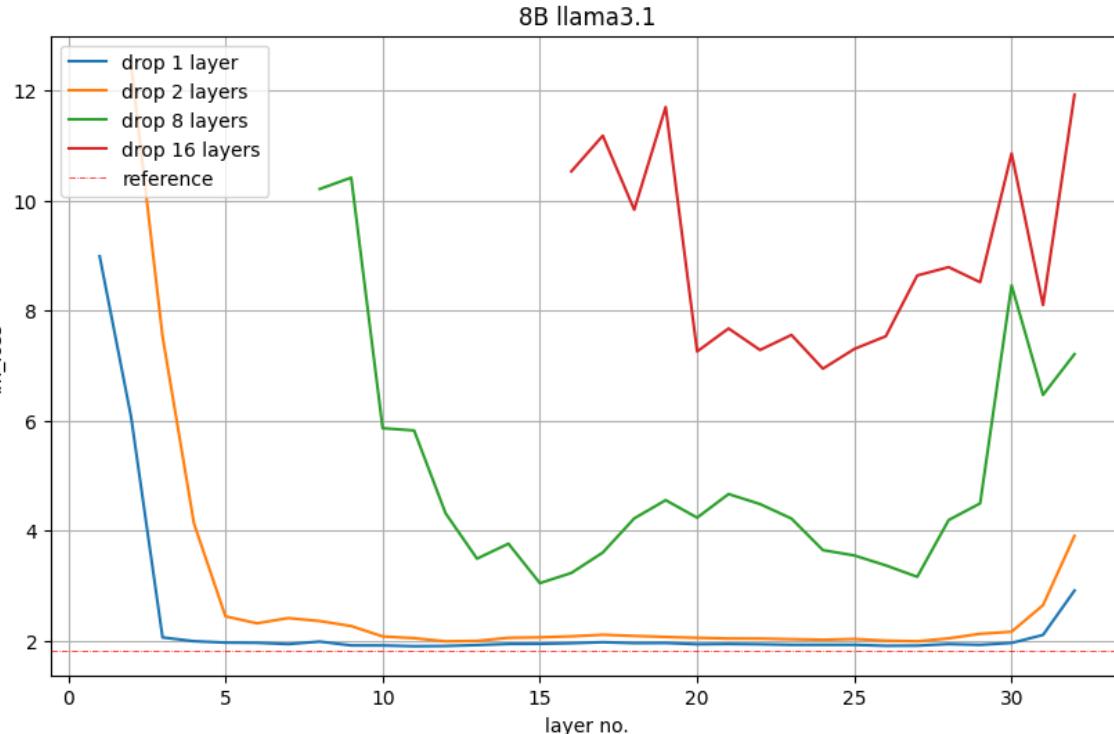
# 結果 (4/4)

		Models						
	Benchmark	Metric	LLMPruner	SliceGPT	LaCo	ShortGPT	Sheared LLaMa	MINITRON
<b>8 Billion</b>	# Parameters		9.8B	9.9B	9.8B	9.8B	-	8.3B
	# Non-Emb. Params		9.5B	9.5B	9.5B	9.5B	-	6.2B
	MMLU(5)	acc	25.2	37.1	45.9	54.7	-	<b>63.8</b>
	hellawag(10)	acc_norm	67.8	55.7	64.4	66.6	-	<b>80.7</b>
	# Parameters		4.8B	4.9B	4.9B	4.9B	2.7B	4.2B
	# Non-Emb. Params		4.5B	4.6B	4.6B	4.6B	2.5B	2.6B
	winogrande (5)	acc	-	-	-	-	64.2	<b>74</b>
	arc_challenge (25)	acc_norm	-	-	-	-	41.2	<b>50.9</b>
	MMLU(5)	acc	23.33	28.92	26.45	43.96	26.4	<b>58.6</b>
	hellawag(10)	acc_norm	56.46	50.27	55.69	53.02	70.8	<b>75</b>
	gsm8k(5)	acc	-	-	-	-	23.96	<b>24.1</b>

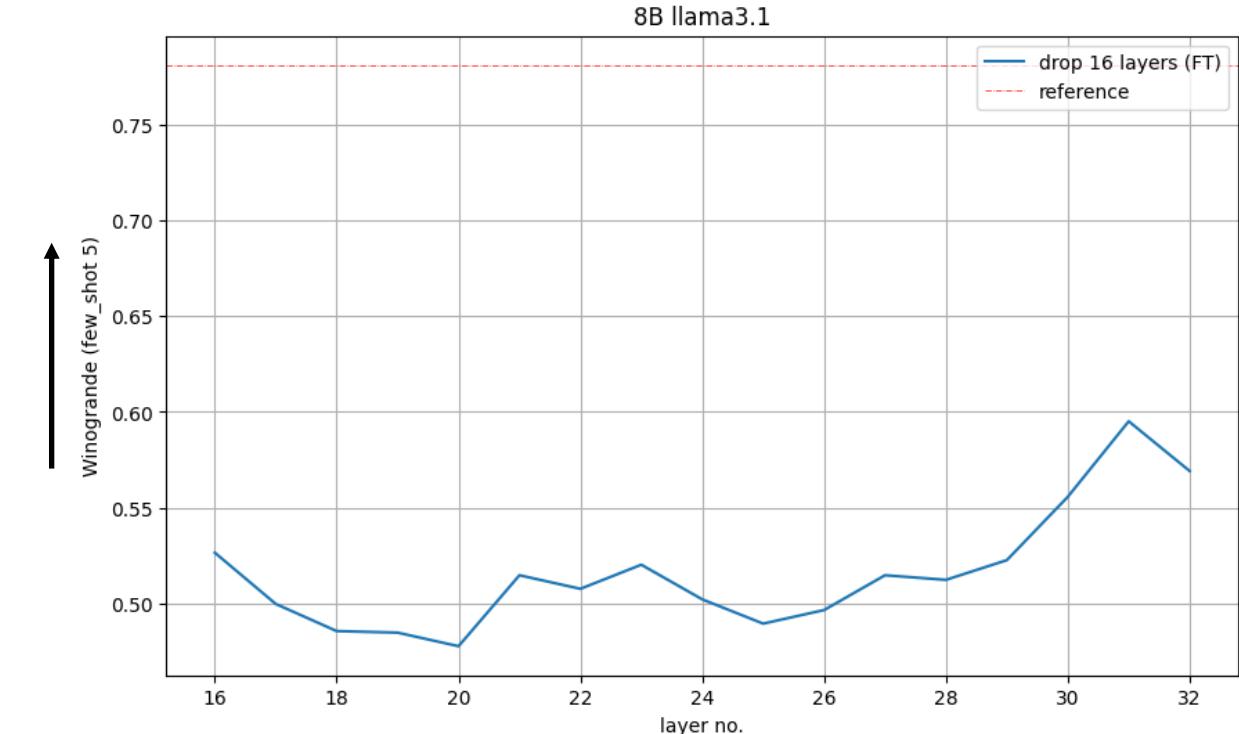
他のPruning済みLLMとの比較  
(Pruning手法同士の比較ではない)

# LLaMA 3.1の場合 | 深さ方向探索のみの結果

- ✓ 重要度はNemotronと同様の計算方法



刈り取る総数とLossの比較



16層枝刈りした時の性能

Lossと精度に相関がみられない

# LLaMA 3.1の場合 | 幅方向探索のPruning

$$F_{\text{head}}^{(i)} = \sum_{\mathbf{B}, \mathbf{S}} \|\text{Attn}(\mathbf{X}\mathbf{W}^{Q,i}, \mathbf{X}\mathbf{W}^{K,i}, \mathbf{X}\mathbf{W}^{V,i})\|_2$$
$$F_{\text{neuron}}^{(i)} = \sum_{\mathbf{B}, \mathbf{S}} \mathbf{X}(\mathbf{W}_1^i)^T, \quad F_{\text{emb}}^{(i)} = \sum_{\mathbf{B}, \mathbf{S}} \text{LN}(\mathbf{X})_i$$

Nemotronと一緒に

- ✓ MLPの中間次元は14,336 → 9,216
- ✓ 隠れ層の次元は4,096 → 3,072
  - ※ 詳細な結果は省略されている

# LLaMA 3.1での場合 | ベンチマーク結果

Benchmark	No. of shots	Metric	Llama-3.1 8B	Minitron 4B	Llama-3.1-Minitron 4B		Phi-2 2.7B	Gemma2 2.6B <sup>†</sup>	Qwen2-1.5B <sup>†</sup>
					Width-pruned	Depth-pruned	Width-pruned		
winogrande	5	acc	0.7727	0.7403*	0.7214	0.7348	0.7400**	0.709	0.662
arc_challenge	25	acc_norm	0.5794	0.5085	0.5256	0.5555**	0.6100*	0.554	0.439
MMLU	5	acc	0.6528	0.5860**	0.5871	0.6053*	0.5749	0.513	0.565
hellaswag	10	acc_norm	0.8180	0.7496	0.7321	0.7606*	0.7524**	0.73	0.666
gsm8k	5	acc	0.4860	0.2411	0.1676	0.4124	0.5500**	0.239	0.585*
truthfulqa	0	mc2	0.4506	0.4288	0.3817	0.4289	0.4400**	-	0.459*
XLSum en (20%)	3	rougeL	0.3005	0.2954*	0.2722	0.2867**	0.0100	-	-
MBPP	0	pass@1	0.4227	0.2817	0.3067	0.324	0.4700*	0.29	0.374**
<b>Training Tokens</b>			15T	94B		1.4T	3T	7T	

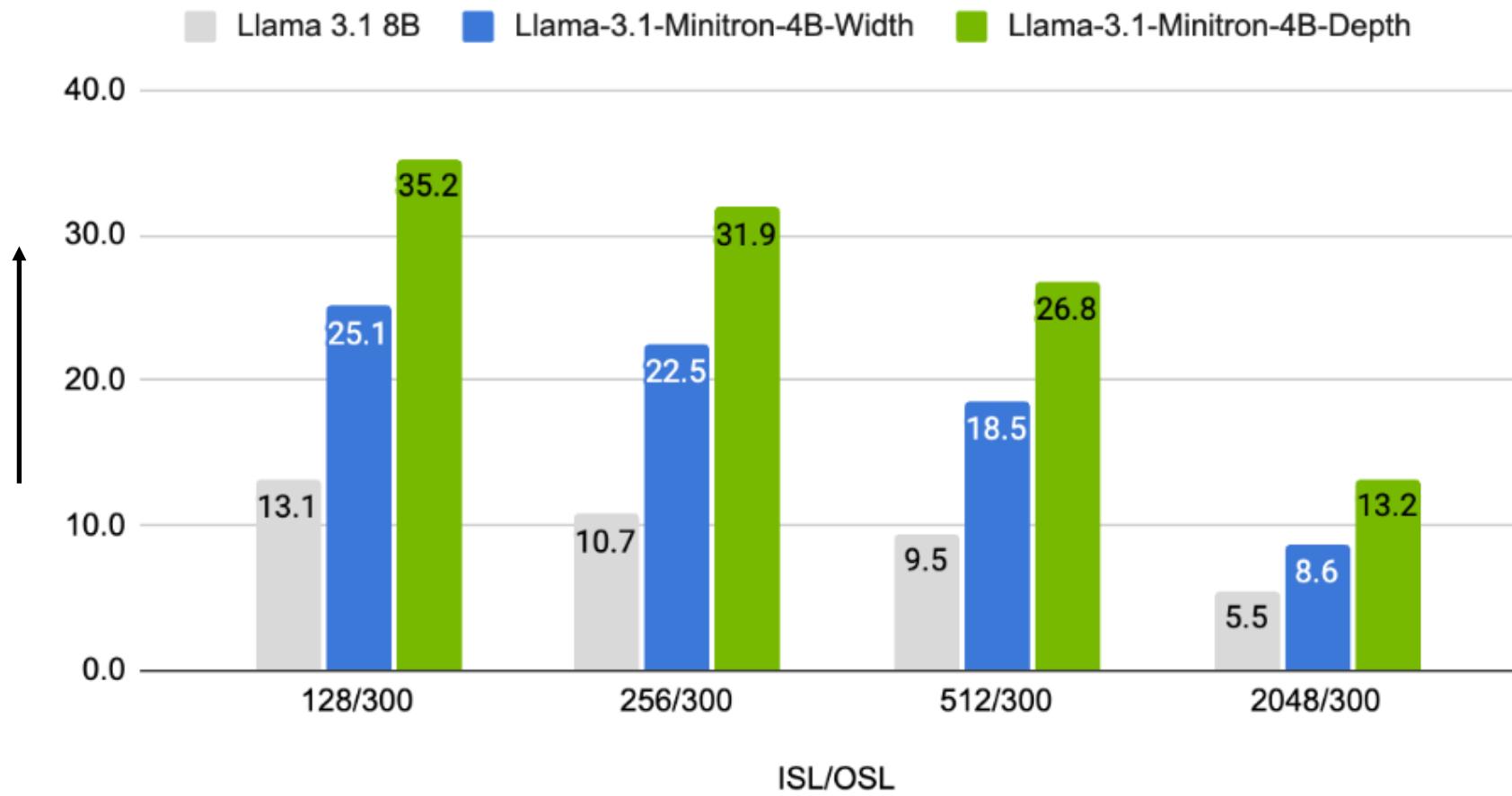
\* : 最高精度

\*\* : 次点

幅方向探索が一部ベンチマークで最高精度

# LLaMA 3.1での場合 | スループット

BF16 Throughput (requests/seconds)



\* : 最高精度  
\*\* : 次点

軽量化による推論速度の改善を確認

# まとめ

## How to Prune and Distill Llama-3.1 8B to an NVIDIA Llama-3.1-Minitron 4B Model

- ✓ PruningとKnowledge DistillationによるLLMの小規模化
  - モデルサイズの削減
    - 15B ⇒ 8B, 8B ⇒ 4B
    - FTデータの削減
      - 8T ⇒ 94B (FTは学習済み地点からの追加学習 なので注意)
  - 推論速度の改善
    - 元の3倍近いスループット

# **Appendix**

# ■ シングルカーネルの誤差近似 | 証明

- ✓ ReLUの非線形性を無視するため、テンソルUの負の要素を0にする

$$g(\mathbf{X}) = \mathbf{V} * (\mathbf{U}^+ * \mathbf{X})$$

- ✓ ReLUの操作が無くなつたことで、2つの畳み込みを1つにまとめられる

$$\mathbf{L}_{i,j,k,1} = \sum_{t=1}^n \mathbf{V}_{1,1,t,1} \cdot \mathbf{U}_{i,j,k,t}^+$$

- ✓ Lの入力は各項目がランダムな変数の合計値

- Pruningで最適な入力を抜き出せば、ランダム初期値で畳み込みを近似可能
  - = Random Subset Sum Problem

# 参考文献

## 宝くじ仮説

- ✓ <https://arxiv.org/abs/1803.03635>
- ✓ <https://arxiv.org/abs/1906.02773>
- ✓ <https://arxiv.org/abs/2111.11146>
- ✓ <https://arxiv.org/abs/1911.13299>
- ✓ <https://arxiv.org/abs/1905.01067>
- ✓ <https://openreview.net/forum?id=Vjki79-619>
- ✓ [https://arxiv.org/abs/1903.01611v1?source=host\\_page](https://arxiv.org/abs/1903.01611v1?source=host_page)-----
- ✓ <https://arxiv.org/abs/1906.03291>

## 枝刈り

- ✓ <https://developer.nvidia.com/blog/how-to-prune-and-distill-llama-3-1-8b-to-an-nvidia-llama-3-1-minitron-4b-model/>
- ✓ <https://arxiv.org/abs/2407.14679>
- ✓ <https://arxiv.org/abs/2308.06767>
- ✓ 某M1の勉強会資料
- ✓ <https://arxiv.org/abs/2006.05525>