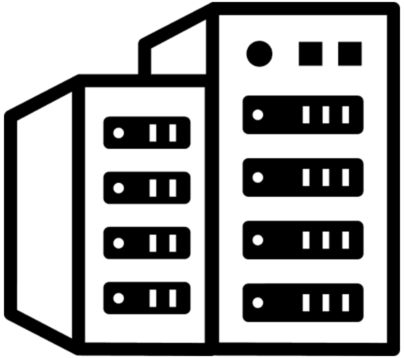
Extracting Effective Subnetworks with Gumbel-Softmax

Robin **Dupont** — Sorbonne Université & Netatmo
Mohammed Amine **Alaoui** — Netatmo
Hichem **Sahbi** — Sorbonne Université
Alice **Lebois** — Netatmo

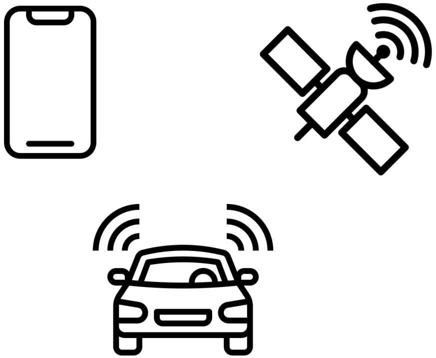
Embedded devices Need Lightweight Neural Networks

Servers



- Powerful 💪
- Handle **full size** models 🧠

Embedded devices



- Limited resources 🤖
- Require **lightweight** models 🪶

How to design lightweight networks ?

Existing network compression methods

- Neural network **distillation**
- Weight **quantization**
- **N**eural **A**rchitecture **S**earch
- **Pruning**
    - Structured
    - **Unstructured**

Existing network compression methods

- Neural network **distillation**
- Weight **quantization**
- **N**eural **A**rchitecture **S**earch
- **Pruning**
  - Structured
  - **Unstructured**

Our method **relies** on **unstructured** pruning

Our method is not a typical pruning method

✅ **Automatic** pruning rate

💡 **No** weight training

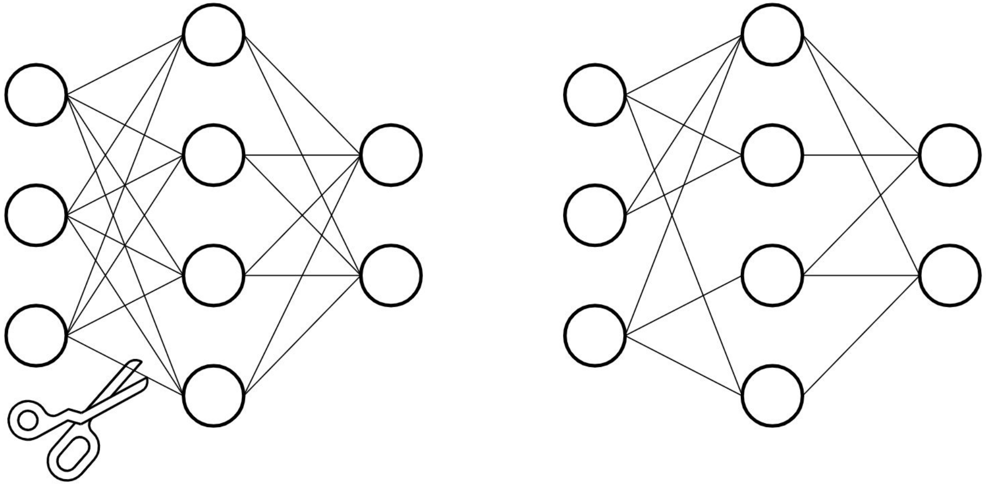
Our method is not a typical pruning method

✅ **Automatic** pruning rate

💡 **No** weight training ➡️ performs **topology selection**

How unstructured weight pruning works ?

Unstructured weight pruning yields lightweights networks



Before pruning

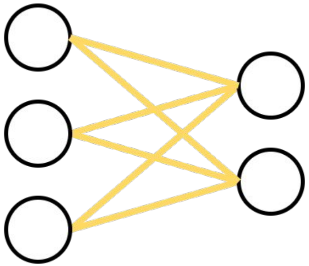
After pruning

- weights removed **individually**
- **Flexible** ✅
- **High sparsity** rate ✅

{Le Cun et al. 1990; Hassibi et al. 1993; Han et al. 2015}

Typical pruning pipelines rely on weight training

Typical pruning pipelines are composed of 3 steps

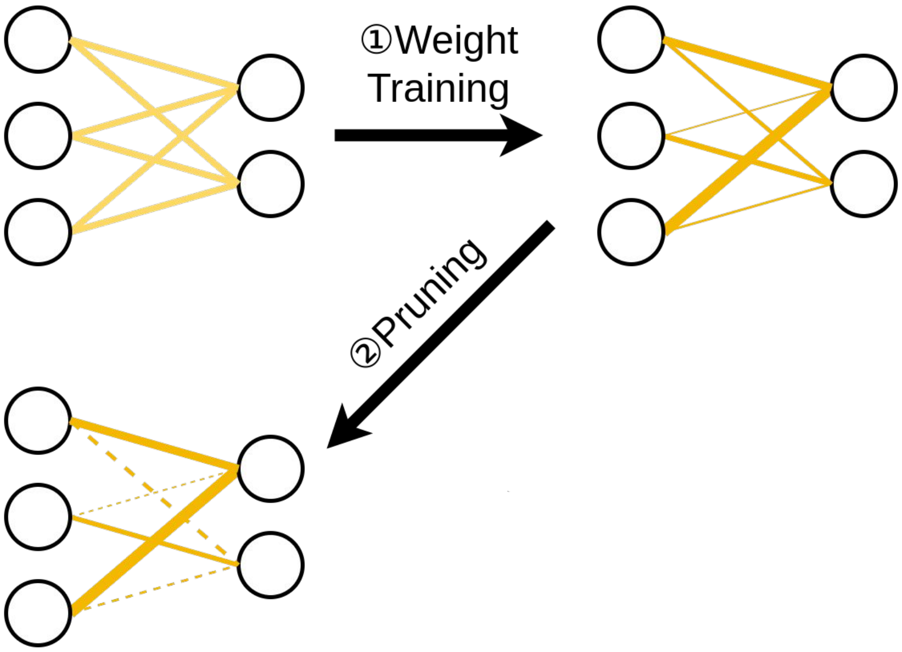


- 3 steps procedure
  train - prune - **fine-tune**
- Pruning **criterion** depends on the **method**
- **Fine-tuning** needed

Typical pruning pipelines are composed of 3 steps

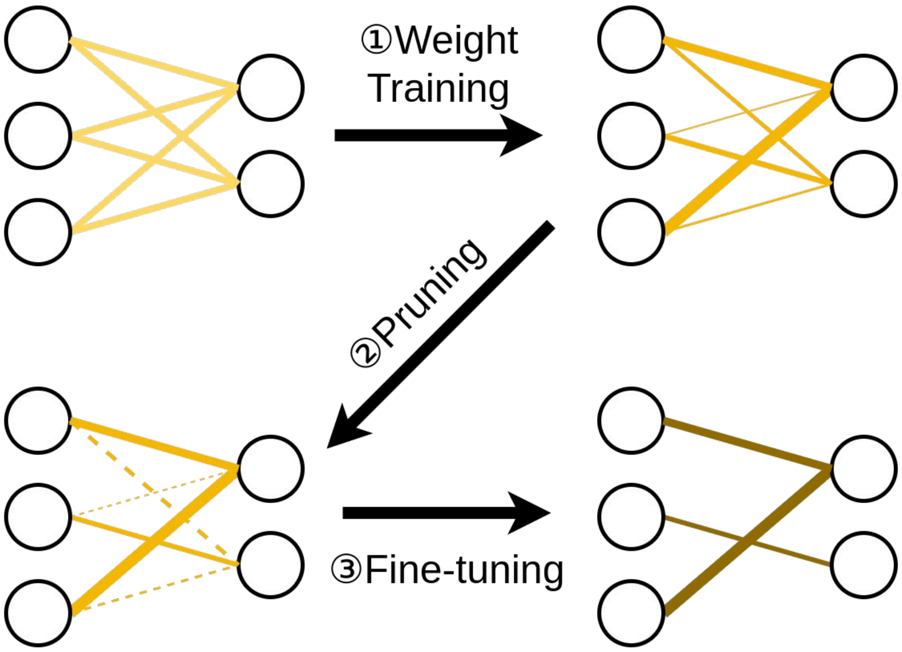
- 3 steps procedure
  train - prune - **fine-tune**
- Pruning **criterion** depends on the **method**
- **Fine-tuning** needed

Typical pruning pipelines are composed of 3 steps



- 3 steps procedure
  train - prune - **fine-tune**
- Pruning **criterion** depends on the **method**
- **Fine-tuning** needed

Typical pruning pipelines are composed of 3 steps

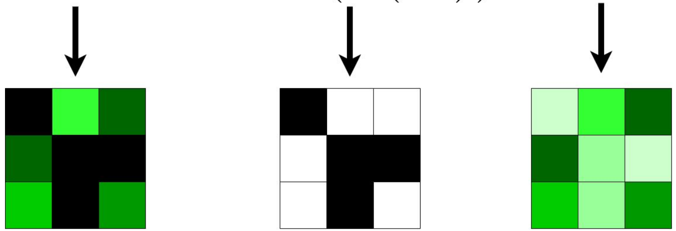


- 3 steps procedure
  train - prune - **fine-tune**
- Pruning **criterion** depends on the **method**
- **Fine-tuning** needed

What if we **do not train** the weights ?

Zhou et al. train the Supermask

$$W \leftarrow \mathrm{Bern}(\sigma(m)) \odot W$$

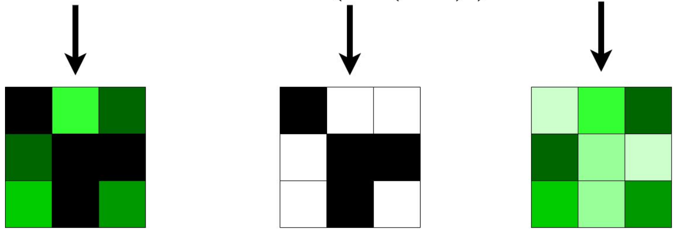


- mask **sampled** from Bernoulli distribution
- optimize **only** *m*

{ Zhou et al. NeurIPS 2019 }

Zhou et al. train the Supermask

$$W \leftarrow \mathrm{Bern}(\sigma(m)) \odot W$$

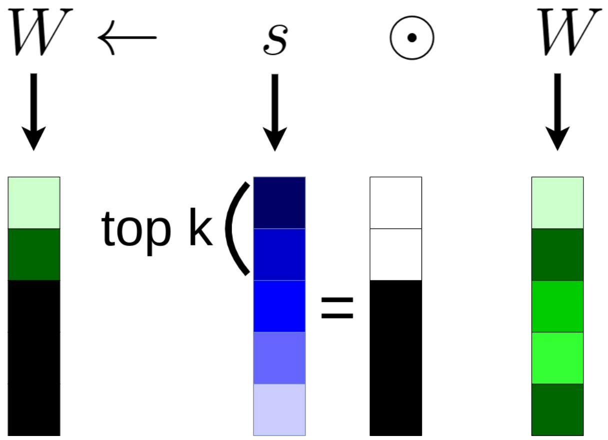


- mask **sampled** from Bernoulli distribution
- optimize **only** *m*

❌ **Basic** mask **parametrization** with Straight Through
❌ **Cumbersome** weight rescaling

{ Zhou et al. NeurIPS 2019 }

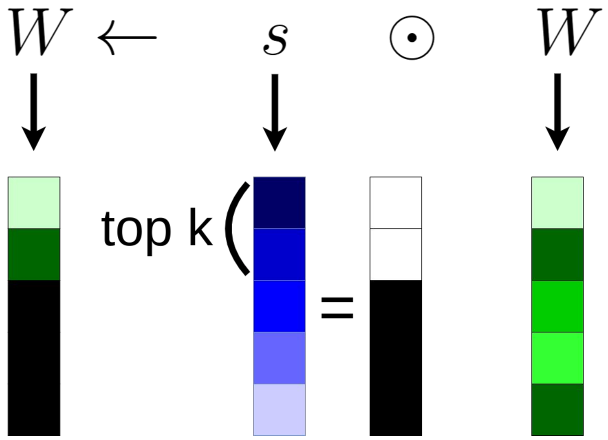
Ramanujan et al. take the top-k weights in each layers



- **top-k** element of *s* are selected
- top-k elements are chosen **per layer**
- k **depends** on the **pruning rate**

{ Ramanujan et al. CVPR 2020 }

Ramanujan et al. take the top-k weights in each layers



- **top-k** element of *s* are selected
- top-k elements are chosen **per layer**
- k **depends** on the **pruning rate**

❌ Pruning rate **has** to be **known in advance**
❌ **Same** pruning rate for **all layer**
❌ Best pruning rate if found by **grid search** ➡️ computationally **expensive** 💰
❌ Best pruning rate **depends on** the network **architecture**

{ Ramanujan et al. CVPR 2020 }

Our Method

Our method characteristics

- We use **Gumbel-Softmax** for differentiable sampling (Jang et al. 2016)
  ✅ better performances than Straight-Through

Our method characteristics

- We use **Gumbel-Softmax** for differentiable sampling (Jang et al. 2016)
  ✅ better performances than Straight-Through

- Pruning rate is **not needed**
  ✅ No computationally intensive grid search

Our method characteristics

- We use **Gumbel-Softmax** for differentiable sampling (Jang et al. 2016)
  ✅ better performances than Straight-Through

- Pruning rate is **not needed**
  ✅ No computationally intensive grid search

- **Learnt** weight rescaling factor
  ✅ Faster inference time and lower overhead

Our method characteristics

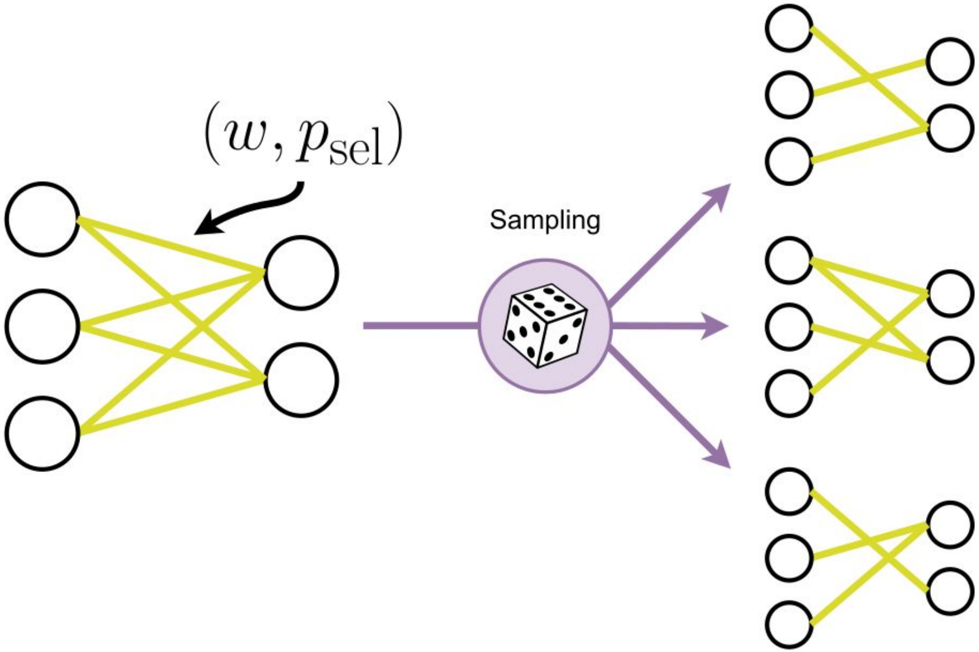
- We use **Gumbel-Softmax** for differentiable sampling (Jang et al. 2016)
  ✅ better performances than Straight-Through

- Pruning rate is **not needed**
  ✅ No computationally intensive grid search

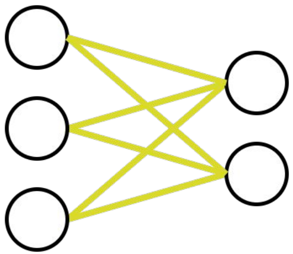
- **Learnt** weight rescaling factor
  ✅ Faster inference time and lower overhead

- **No** weight training
  💡 Topology selection only

Each weight as a probability of being selected

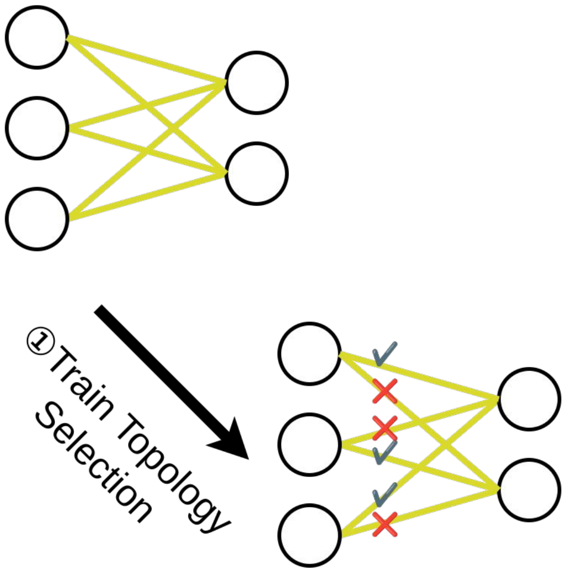


Our method performs topology selection only



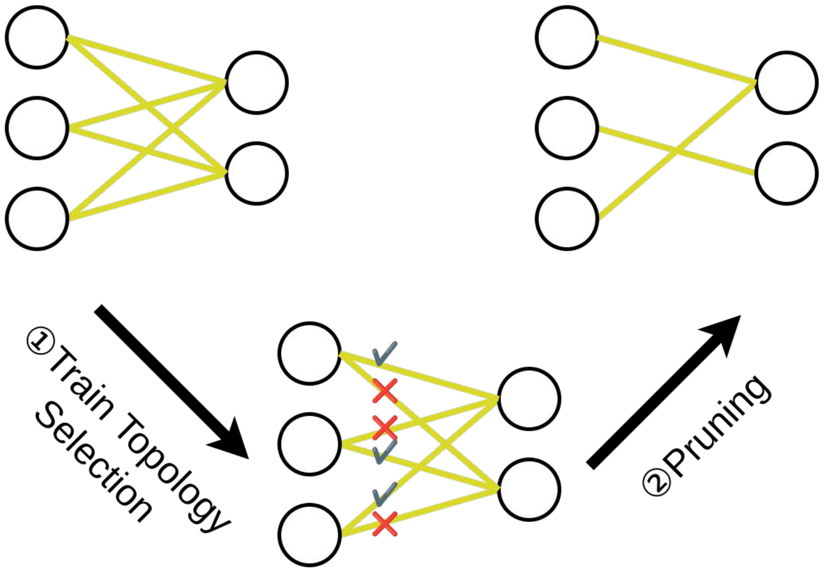
- No weight **training** or **fine-tuning** ⚠️
- **Topology selection** only

Our method performs topology selection only



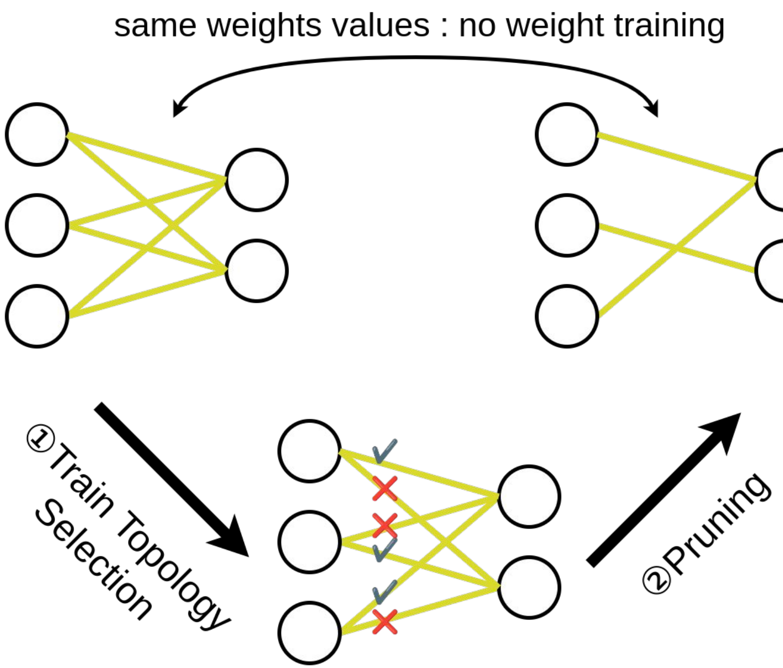
- No weight **training** or **fine-tuning** ⚠️
- **Topology selection** only

Our method performs topology selection only



- No weight **training** or **fine-tuning** ⚠️
- **Topology selection** only

Our method performs topology selection only



- No weight **training** or **fine-tuning** ⚠️
- **Topology selection** only

Topology are selected with a stochastic mask

Layer equation :

$$\mathbf{z}_\ell = g_\ell((\boldsymbol{m}_\ell \odot \boldsymbol{w}_\ell) \otimes \mathbf{z}_{\ell-1})$$

binary masks tensor

weights tensor

Topology are selected with a stochastic mask

Layer equation : $$\mathbf{z}_\ell = g_\ell((\boldsymbol{m}_\ell \odot \boldsymbol{w}_\ell) \otimes \mathbf{z}_{\ell-1})$$

binary masks tensor

weights tensor

each coefficient follows Bernoulli distribution : $m \sim \mathcal{B}(p_{\mathrm{sel}})$

Probability for a **weight** to **be selected**

Topology are selected with a stochastic mask

Layer equation : $$\mathbf{z}_\ell = g_\ell((\boldsymbol{m}_\ell \odot \boldsymbol{w}_\ell) \otimes \mathbf{z}_{\ell-1})$$

binary masks tensor

weights tensor

each coefficient follows Bernoulli distribution : $m \sim \mathcal{B}(p_{\text{sel}})$

Sampling is **not differentiable** ⚠️

Probability for a **weight** to **be selected**

Probability is reparametrized

Probability reparametrization : $$p_{\text{sel}} = \sigma(\hat{m})$$

Learnt variable

Sigmoid ensures $0 \leq p_{\text{sel}} \leq 1$

Naive Straight Through Gumbel-Softmax is flawed

Probability reparametrization : $$p_{\text{sel}} = \sigma(\hat{m})$$

Naive **S**traight **T**hrough **G**umbel-**S**oftmax formulation :

$$m = \text{STGS}\left(\begin{bmatrix} \log(\sigma(\hat{m})) \\ \log(1-\sigma(\hat{m})) \end{bmatrix}\right)$$

Naive Straight Through Gumbel-Softmax is flawed

Probability reparametrization : $$p_{\text{sel}} = \sigma(\hat{m})$$

Naive **S**traight **T**hrough **G**umbel-**S**oftmax formulation :

$$m = \text{STGS}\left(\begin{bmatrix} \log(\sigma(\hat{m})) \\ \log(1-\sigma(\hat{m})) \end{bmatrix}\right)$$

Combination of log and exponential functions :

❌ Numerical **instabilities**
❌ Computationally **intensive**

ASLP is simpler and resolves issues

Probability reparametrization : $$p_{\mathrm{sel}} = \sigma(\hat{m})$$

Our formulation **Arbitrarily Shifted Log Parameterization** (ASLP)

$$m = \mathrm{STGS}\left(\begin{bmatrix} \hat{m} \\ 0 \end{bmatrix}\right)$$

ASLP is simpler and resolves issues

Probability reparametrization : $$p_{\text{sel}} = \sigma(\hat{m})$$

Our formulation **Arbitrarily Shifted Log Parameterization** (ASLP)

$$m = \text{STGS}\left(\begin{bmatrix} \hat{m} \\ 0 \end{bmatrix}\right)$$

✅ Numerically **stable**
✅ **Less** computationally intensive

ASLP formulation implies the same parametrization for P_{sel}

Our formulation : $$m = \mathrm{STGS}\left(\begin{bmatrix} \hat{m} \\ 0 \end{bmatrix}\right)$$

Arbitrary unknown constant that shifts log probabilities

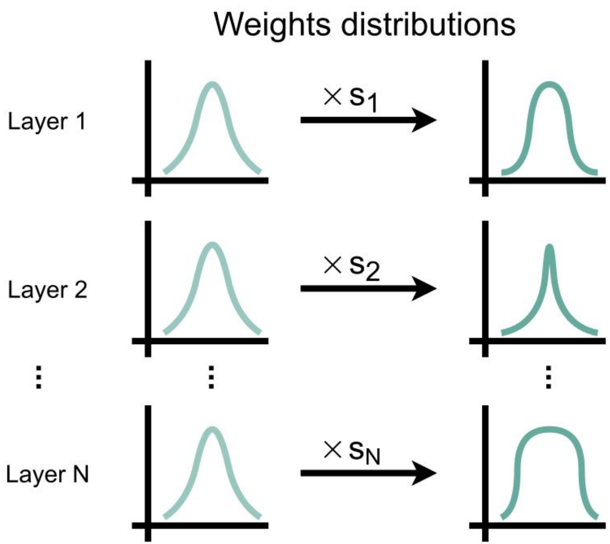
$$\begin{bmatrix} \hat{m} \\ 0 \end{bmatrix} = \begin{bmatrix} \log(p_{\mathrm{sel}}) \ + \ c \\ \log(1 - p_{\mathrm{sel}}) \ + \ c \end{bmatrix} \Rightarrow p_{\mathrm{sel}} = \sigma(\hat{m})$$

💡 Adding a constant **does not change** the result of STGS

Same reparametrization

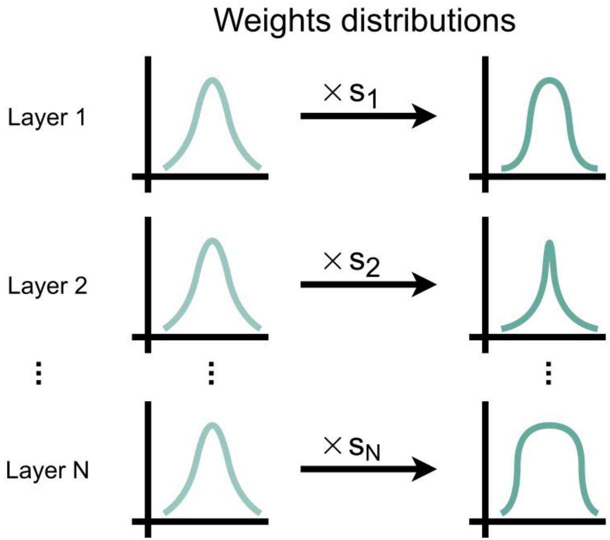
Pruning weights affects the signal propagation dynamic

Smart rescale is a simpler and useful weight rescaling



- Scaling learnt per layer
- **Mitigates** the change of **variance** due to **pruning**
- Improves **performances**
- **Reduces** number of epochs needed for **convergence**

Smart rescale is a simpler and useful weight rescaling

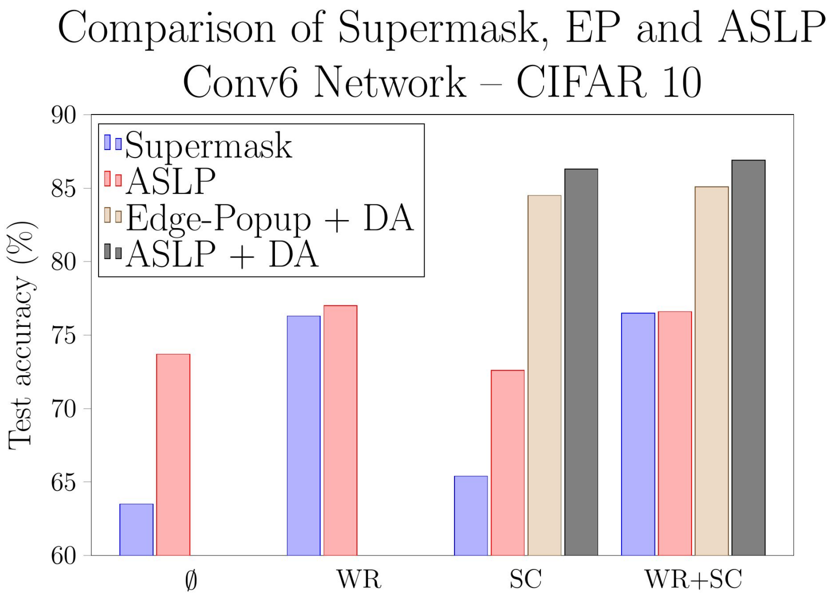


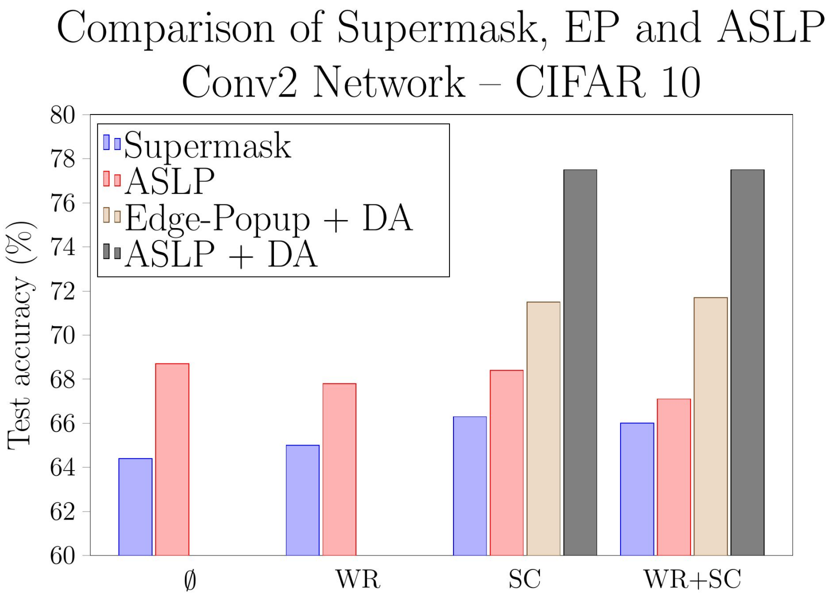
- Scaling learnt per layer
- **Mitigates** the change of **variance** due to **pruning**
- Improves **performances**
- **Reduces** number of epochs needed for **convergence**

✅ **Less computationally intensive** than Zhou et al. weight rescaling

Results

Results : Our method performs better on various cases



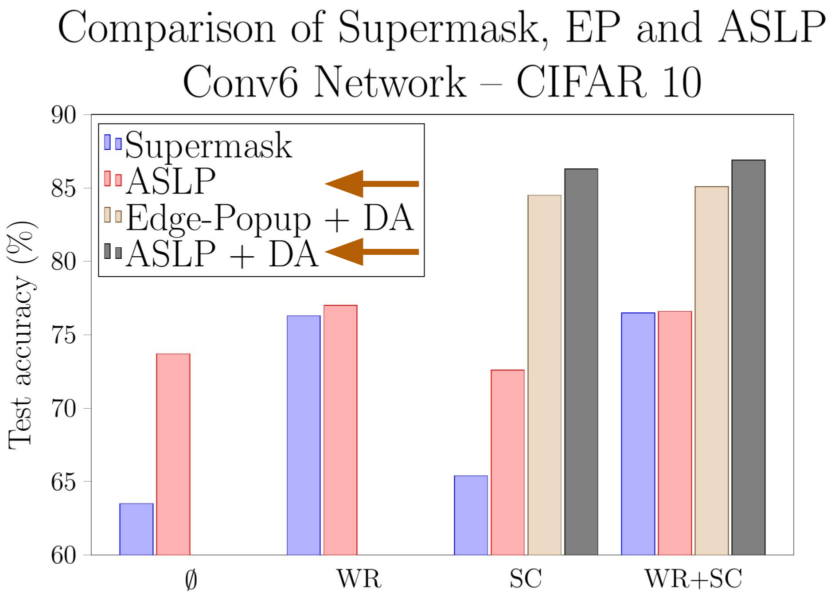


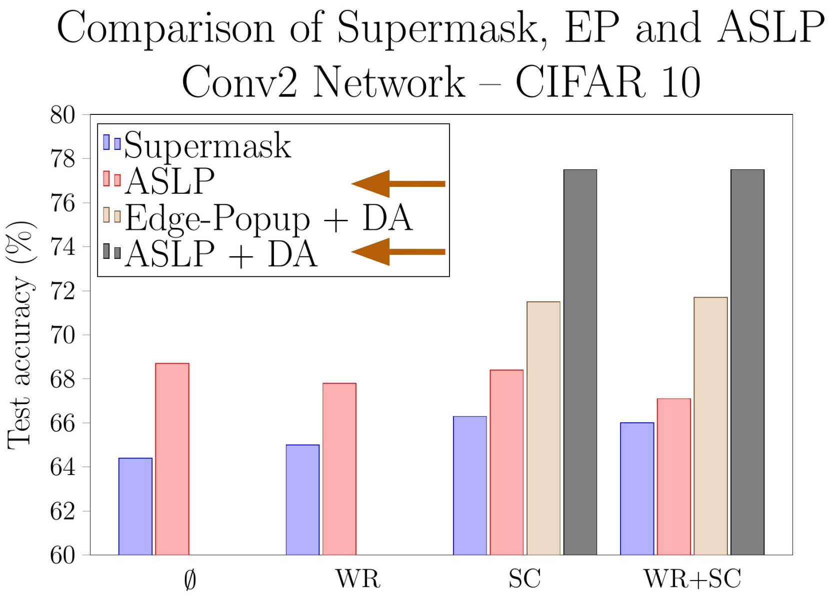
WR = Weight Rescale, SC = Signed Constant, DA = Data Augmentation

EP (Edge-Popup)[1], Supermask[2]

[1] H. Zhou, et al. "Deconstructing lottery tickets: Zeros, signs, and the supermask," in NeurIPS, 2019
[2] V. Ramanujan, et al. "What's hidden in a randomly weighted neural network?," in CVPR, 2020

Results : Our method performs better on various cases





WR = Weight Rescale, SC = Signed Constant, DA = Data Augmentation

EP (Edge-Popup)[1], Supermask[2]

[1] H. Zhou, et al. "Deconstructing lottery tickets: Zeros, signs, and the supermask," in NeurIPS, 2019
[2] V. Ramanujan, et al. "What's hidden in a randomly weighted neural network?," in CVPR, 2020

Smart Rescale is faster than DWR[1] and accelerate convergence

- Smart rescale overhead **0.13s**
- DWR[1] overhead **0.2s**

[1] H. Zhou, et al. "Deconstructing lottery tickets: Zeros, signs, and the supermask," in NeurIPS, 2019

Smart Rescale is faster than DWR[1] and accelerate convergence

- Smart rescale overhead **0.13s**

✅ SR overhead **35% faster**

- DWR[1] overhead **0.2s**

[1] H. Zhou, et al. "Deconstructing lottery tickets: Zeros, signs, and the supermask," in NeurIPS, 2019

Smart Rescale is faster than DWR[1] and accelerate convergence

- Smart rescale overhead **0.13s**

✅ SR overhead **35% faster**

- DWR[1] overhead **0.2s**

- # Epochs reduction :
    - **8.2%** (Conv2)
    - **19.7%** (Conv4)
    - **14.0%** (Conv6)

[1] H. Zhou, et al. "Deconstructing lottery tickets: Zeros, signs, and the supermask," in NeurIPS, 2019

Results : On CIFAR100 ASLP performs better on most cases

	Conv2	Conv4	Conv6
EP	40.9	51.1	**53.2**
ASLP	**43.4**	**51.7**	52.8

Table 1: Edge Popup and ASLP on CIFAR100

Results with Weight Rescale (WR) and Signed Constant (SC)

V. Ramanujan, et al. "What's hidden in a randomly weighted neural network?," in CVPR, 2020

Sum Up

Sum Up

- Prune **untrained** networks ➡️ **topology selection** only
- **Gumbel Softmax** ➡️ differentiable sampling
- **ASLP** :
  - simpler formulation
  - ✅ **less** computationally intensive
  - ✅ numerically **stable**
- **Smart Rescale** :
  - ✅ **improves** performances
  - ⏱️ **reduces** number of **epochs**
- Our method yields **lightweight** networks 🪶, **without weight training.**

A few perspectives

- Test ASLP on other **network architectures** and **datasets**
- **Reduce** training time
- Test ASLP on **other context and applications**

Thank you! 👍

Code available at:
`github.com/n0ciple/aslp`

Robin **Dupont** — Sorbonne Université & Netatmo
Mohammed Amine **Alaoui** — Netatmo
Hichem **Sahbi** — Sorbonne Université
Alice **Lebois** — Netatmo