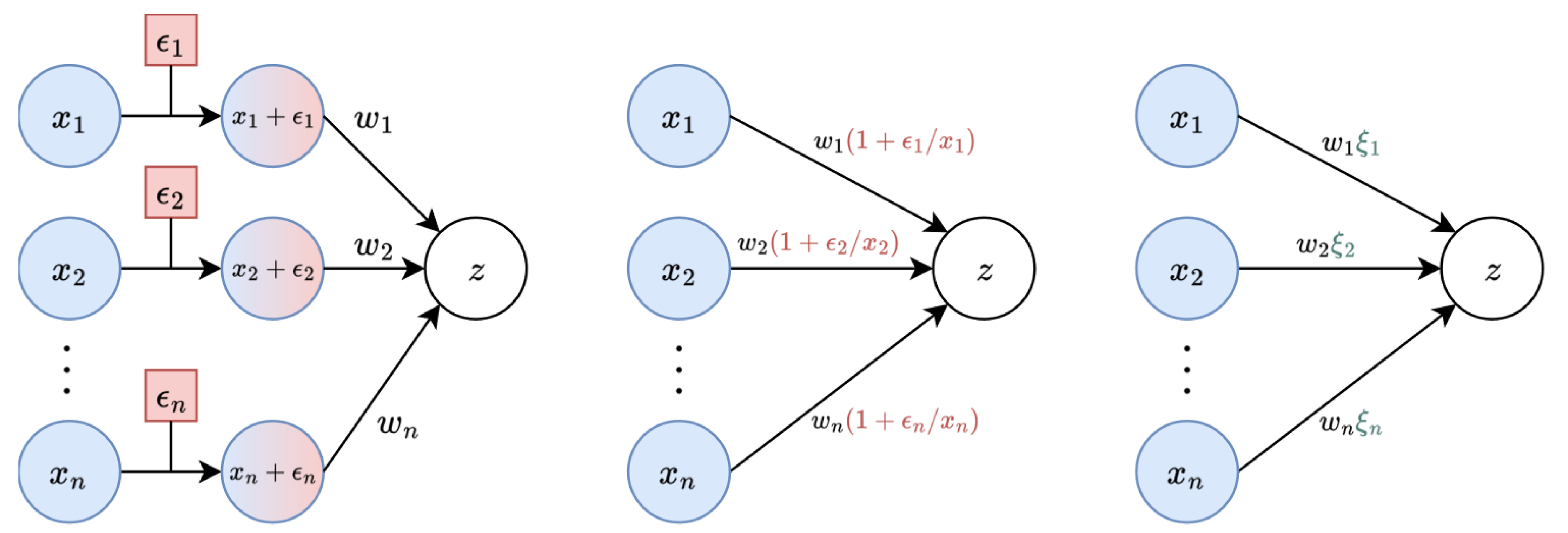
Improving robustness to corruptions with multiplicative weight perturbations

Trung Trinh[1], Markus Heinonen[1], Luigi Acerbi[2], Samuel Kaski[1,3]
[1]Aalto University, [2]Helsinki University, [3]University of Manchester
{trung.trinh, markus.o.heinonen, samuel.kaski}@aalto.fi, luigi.acerbi@helsinki.fi

Scan here for project website and code

Main takeaway

We show that multiplying each individual weight of a neural network with its own Gaussian random variable during training will enhance the model's robustness to distribution shift.



(a) $z = \mathbf{w}^\top(\mathbf{x}+\boldsymbol{\epsilon})$ (b) $z = (\mathbf{w}\circ(\mathbf{1}+\boldsymbol{\epsilon}/\mathbf{x}))^\top\mathbf{x}$ (c) $z = (\mathbf{w}\circ\boldsymbol{\xi})^\top\mathbf{x}, \boldsymbol{\xi}\sim p(\boldsymbol{\xi})$

Theorem

Given a dataset $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N \subset \mathcal{X}\times\mathcal{Y}$, a corruption $\mathrm{g}: \mathcal{X}\to\mathcal{X}$, neural network weights $\boldsymbol{\omega}\in\mathcal{W}$, and a loss function $\mathcal{L}$. Under some weak assumptions about g and $\mathcal{L}$, there exists a multiplicative weight perturbation $\boldsymbol{\xi}_{\mathrm{g}}\in\mathcal{W}$ and a constant $C_{\mathrm{g}}>0$ such that:

$$\underbrace{\mathcal{L}(\boldsymbol{\omega};\mathrm{g}(\mathcal{S}))}_{\text{Loss under corruption}} \leq \underbrace{\mathcal{L}(\boldsymbol{\omega}\circ\boldsymbol{\xi}_{\mathrm{g}};\mathcal{S})}_{\text{Loss under perturbation}} + \underbrace{\frac{C_{\mathrm{g}}}{2}\|\boldsymbol{\omega}\|_F^2}_{L_2\text{ regularization}}$$

where $\mathrm{g}(\mathcal{S}) = \{(\mathrm{g}(\mathbf{x}_i), y_i)\}_{i=1}^N$.

Implications: The multiplicative weight perturbations simulate input corruptions during training, making the model robust to these simulated corruptions, which could also improve its robustness to real world corruptions.

Previous works using multiplicative weight perturbations

Dropout [1]: multiplies all weights connecting to the same node with a Bernoulli random variable.
Adaptive Sharpness aware minimization (ASAM) [2]: originally proposed as improved version of SAM [3], we show that ASAM can be interpreted as optimizing neural networks under adversarial multiplicative weight perturbations, which explains its higher performance than SAM under corruptions.

Proposed method: Data Augmentation via Multiplicative Perturbations (DAMP)

DAMP is an efficient training method that perturbs weights using multiplicative Gaussian random variable during training by minimizing the following loss function:

$$\mathcal{L}_{\mathrm{DAMP}}(\boldsymbol{\omega};\mathcal{S}) := \mathbb{E}_{\boldsymbol{\xi}\sim\mathcal{N}(\mathbf{1},\sigma^2\mathbf{I})}\left[\mathcal{L}(\boldsymbol{\omega}\circ\boldsymbol{\xi};\mathcal{S})\right] + \frac{\lambda}{2}\|\boldsymbol{\omega}\|_F^2$$

To efficiently estimate the expectation, DAMP splits the training batch evenly into *M* sub-batches, then samples a weight perturbation for each sub-batch to calculate the sub-batch gradient, and finally averages over all sub-batch gradients to obtain the final gradient. Therefore, DAMP is suitable for data parallelism in multi-GPU training.

Experiment results

Table 1: **DAMP surpasses the baselines on corrupted images in most cases.** We report the predictive errors (lower is better) averaged over 3 seeds for the ResNet50 / ImageNet experiments. For each level of severity in ImageNet-C and ImageNet-$\bar{\text{C}}$, we report the average error over all corruption types. We use 90 epochs and the basic Inception-style preprocessing for all experiments.

Method	ImageNet	ImageNet-C					ImageNet-$\bar{\text{C}}$				
		1	2	3	4	5	1	2	3	4	5
SGD	$23.6_{\pm0.2}$	$40.2_{\pm0.1}$	$51.0_{\pm0.2}$	$61.1_{\pm0.3}$	$73.5_{\pm0.2}$	$82.9_{\pm<0.1}$	$45.7_{\pm0.1}$	$54.7_{\pm0.1}$	$61.8_{\pm<0.1}$	$70.3_{\pm0.1}$	$75.7_{\pm0.1}$
DAMP	$23.8_{\pm<0.1}$	$38.2_{\pm<0.1}$	$48.1_{\pm<0.1}$	$57.4_{\pm0.2}$	$\mathbf{69.5_{\pm0.3}}$	$\mathbf{79.9_{\pm0.2}}$	$\mathbf{42.4_{\pm0.1}}$	$\mathbf{51.2_{\pm0.1}}$	$\mathbf{58.6_{\pm<0.1}}$	$\mathbf{67.8_{\pm0.1}}$	$\mathbf{73.4_{\pm0.1}}$
SAM	$23.1_{\pm0.1}$	$38.7_{\pm0.3}$	$49.1_{\pm0.3}$	$59.1_{\pm0.3}$	$71.6_{\pm0.1}$	$81.7_{\pm0.1}$	$44.5_{\pm<0.1}$	$53.5_{\pm0.1}$	$60.8_{\pm<0.1}$	$69.4_{\pm<0.1}$	$75.1_{\pm0.1}$
ASAM	$\mathbf{22.8_{\pm0.1}}$	$\mathbf{37.4_{\pm0.1}}$	$\mathbf{47.5_{\pm<0.1}}$	$\mathbf{57.3_{\pm0.1}}$	$70.0_{\pm0.1}$	$80.7_{\pm0.1}$	$42.5_{\pm0.1}$	$51.4_{\pm0.1}$	$59.0_{\pm0.1}$	$68.2_{\pm0.1}$	$74.0_{\pm<0.1}$

Table B: **ViT-S16 / ImageNet (IN) with basic Inception-style data augmentations**. This table extends the results of Table 2 in the paper. We further evaluate the models on IN-A, IN-D, IN-Sketch, IN-Drawing, IN-Cartoon and adversarial examples generated by FGSM. For IN-C and IN-$\overline{\text{C}}$, we report the results averaged over all corruption types and severity levels. For FGSM, we use $\epsilon = 2/224$. We also report the runtime of each experiment, showing that SAM and ASAM take twice as long to run than DAMP and Dropout given the same number of epochs. The `Avg` column displays the average of all previous columns except `IN-Clean`. Overall, DAMP produces the most robust model on average.

Method	#Epochs	Runtime	Error (%) ↓									
			IN-Clean	FGSM	IN-A	IN-C	IN-$\overline{\text{C}}$	IN-Cartoon	IN-D	IN-Drawing	IN-Sketch	Avg
Dropout	100	20.6h	28.55	93.47	93.44	65.87	64.52	50.37	91.15	79.62	88.06	78.31
	200	41.1h	28.74	90.95	93.33	66.90	64.83	51.23	92.56	81.24	87.99	78.63
DAMP	100	20.7h	25.50	92.76	92.92	57.85	57.02	44.78	88.79	69.92	83.16	73.40
	200	41.1h	**23.75**	**84.33**	**90.56**	55.58	**55.58**	41.06	**87.87**	68.36	81.82	**70.65**
SAM	100	41h	23.91	87.61	93.96	55.56	55.93	42.53	88.23	69.53	81.86	71.90
ASAM	100	41.1h	24.01	85.85	92.99	**55.13**	55.64	**40.74**	89.03	**67.80**	**81.47**	71.08

Table D: **ViT / ImageNet (IN) with MixUp and RandAugment**. We train ViT-S16 and ViT-B16 on ImageNet from scratch with advanced data augmentations (DAs). We evaluate the models on IN-C, IN-$\overline{\text{C}}$, IN-A, IN-D, IN-Sketch, IN-Drawing, IN-Cartoon and adversarial examples generated by FGSM. For FGSM, we use $\epsilon = 2/224$. For IN-C and IN-$\overline{\text{C}}$, we report the results averaged over all corruption types and severity levels. The `Avg` column displays the average of all previous columns except `IN-Clean`. These results indicate that: (i) DAMP can be combined with modern DA techniques to further enhance robustness; (ii) DAMP is capable of training large models like ViT-B16; (iii) given the same amount of training time, it is better to train a large model (ViT-B16) using DAMP than to train a smaller model (ViT-S16) using SAM/ASAM.

Model	Method	#Epochs	Runtime	Error (%) ↓									
				IN-Clean	FGSM	IN-A	IN-C	IN-$\overline{\text{C}}$	IN-Cartoon	IN-D	IN-Drawing	IN-Sketch	Avg
ViT-S16	Dropout	500	111h	20.25	62.45	40.85	84.29	44.72	34.35	86.59	56.31	71.03	60.07
	DAMP	500	111h	**20.09**	59.87	**39.30**	**83.12**	**43.18**	34.01	**84.74**	**54.16**	**68.03**	**58.30**
	SAM	300	123h	20.17	59.92	40.05	83.91	44.04	34.34	85.99	55.63	70.85	59.34
	ASAM	300	123h	20.38	**59.38**	39.44	83.64	43.41	**33.82**	85.41	54.43	69.13	58.58
ViT-B16	Dropout	275	123h	20.41	56.43	39.14	82.85	43.82	33.13	87.72	56.15	71.36	58.83
	DAMP	275	124h	**19.36**	**55.20**	**37.77**	**80.49**	**41.67**	**31.63**	**87.06**	**52.32**	**67.91**	**56.76**

[1] Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR 2014.
[2] Kwon et al. ASAM: Adaptive Sharpness-Aware Minimization for Scale-Invariant Learning of Deep Neural Networks. ICML 2021.
[3] Foret et al. Sharpness-Aware Minimization for Efficiently Improving Generalization. ICLR 2021.