Published as a conference paper at ICLR 2021

SHARPNESS-AWARE MINIMIZATION FOR EFFICIENTLY IMPROVING GENERALIZATION

Pierre Foret *
Google Research
pierre.pforet@gmail.com

Ariel Kleiner
Google Research
akleiner@gmail.com

Hossein Mobahi
Google Research
hmobahi@google.com

Behnam Neyshabur
Blueshift, Alphabet
neyshabur@google.com

https://arxiv.org/pdf/2010.01412.pdf

12.17.2021

What is Generalization?

- “How well trained model performs on new, unseen data”
  - Our test set should model this

What is Generalization?

- Do two models with the **same** architecture, **same** train loss / accuracy, but **different** weights perform the same on unseen data?

What is Generalization?

NO!

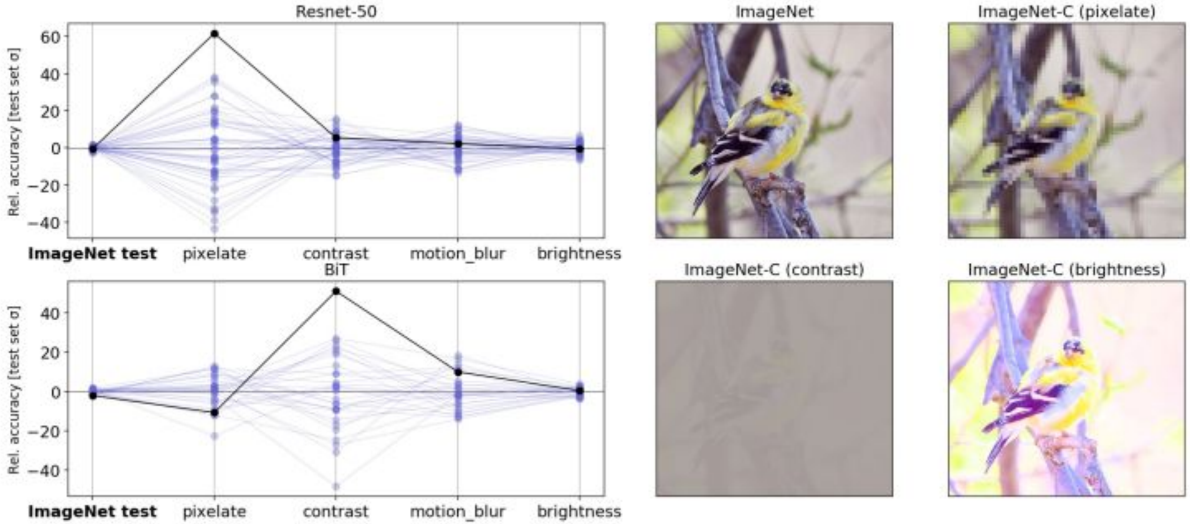
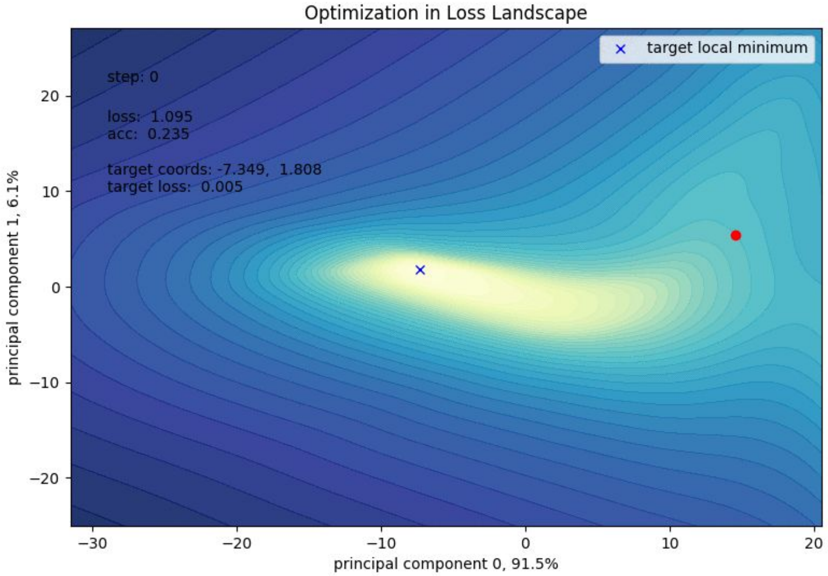


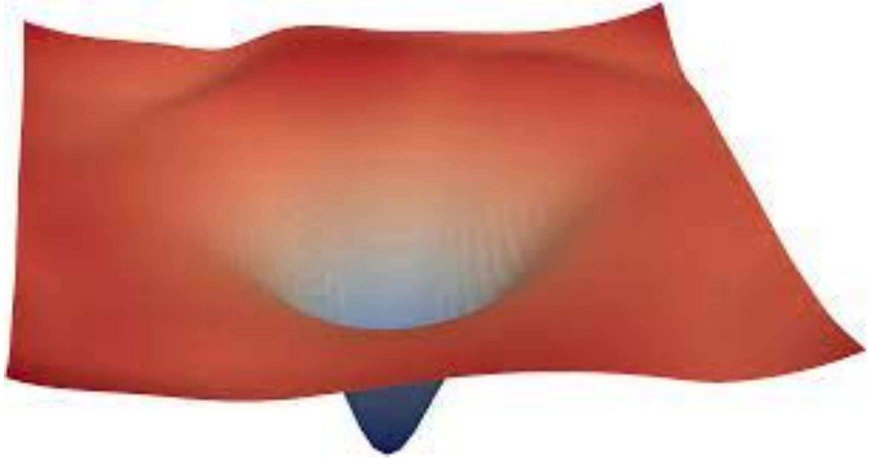
Figure 4: **Image classification model performance on stress tests is sensitive to random initialization in ways that are not apparent in iid evaluation. (Top Left)** Parallel axis plot showing variation in accuracy between identical, randomly initialized ResNet 50 models on several ImageNet-C tasks at corruption strength 5. Each line corresponds to a particular model in the ensemble; each each parallel axis shows deviation from the ensemble mean in accuracy, scaled by the standard deviation of accuracies on the "clean" ImageNet test set. On some tasks, variation in performance is orders of magnitude larger than on the standard test set. **(Right)** Example image from the standard ImageNet test set, with corrupted versions from the ImageNet-C benchmark.

Underspecification Presents Challenges for Credibility in Modern Machine Learning
https://arxiv.org/pdf/2011.03395.pdf

What is a Loss Landscape?

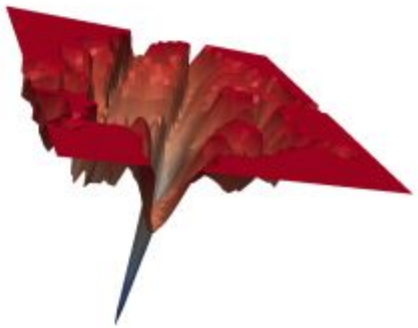
- A model can have different weights that determine its train loss
  - At initialization - high loss
  - After training - low loss (hopefully)

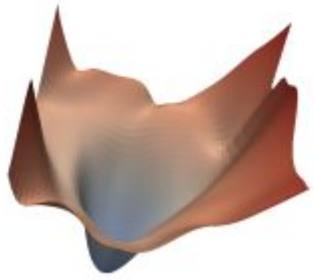




What is a Loss Landscape?

- The local minimas of the loss landscape are basins
  - They can be flat or sharp
- Generally flat landscapes are associated with better generalization





http://proceedings.mlr.press/v80/dziugaite18a/dziugaite18a.pdf
https://arxiv.org/pdf/1611.01838.pdf
https://proceedings.neurips.cc/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf

Kitware

Sharpness-Aware Minimization (SAM)

- SGD finds weights with low loss
  - Minimizes loss
- SAM finds a neighborhood of weights with loss (aka a flat basin)
  - Minimizes loss and loss sharpness

Sharpness-Aware Minimization (SAM)

TL;DR - At each iteration, first climb to local maxima of neighborhood, and then calculate loss

$$\min_{\boldsymbol{w}} L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) + \lambda ||\boldsymbol{w}||_2^2 \quad \text{where} \quad L_{\mathcal{S}}^{SAM}(\boldsymbol{w}) \triangleq \max_{||\boldsymbol{\epsilon}||_p \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}),$$

$$\boldsymbol{\epsilon}^*(\boldsymbol{w}) \triangleq \arg\max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} L_{\mathcal{S}}(\boldsymbol{w} + \boldsymbol{\epsilon})$$

Input: Training set $\mathcal{S} \triangleq \cup_{i=1}^n \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$, Loss function $l : \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_+$, Batch size b, Step size $\eta > 0$, Neighborhood size $\rho > 0$.
Output: Model trained with SAM
Initialize weights $\boldsymbol{w}_0$, $t = 0$;
while *not converged* **do**
  Sample batch $\mathcal{B} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), ...(\boldsymbol{x}_b, \boldsymbol{y}_b)\}$;
  Compute gradient $\nabla_{\boldsymbol{w}} L_{\mathcal{B}}(\boldsymbol{w})$ of the batch's training loss;
  Compute $\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})$ per equation 2;
  Compute gradient approximation for the SAM objective (equation 3): $\boldsymbol{g} = \nabla_w L_{\mathcal{B}}(\boldsymbol{w})|_{\boldsymbol{w}+\hat{\boldsymbol{\epsilon}}(\boldsymbol{w})}$;
  Update weights: $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}$;
  $t = t + 1$;
end
return $\boldsymbol{w}_t$

Algorithm 1: SAM algorithm

Improves Performance across Models, Datasets, & Training Procedures

		CIFAR-10		CIFAR-100	
Model	Augmentation	SAM	SGD	SAM	SGD
WRN-28-10 (200 epochs)	Basic	**2.7**$_{\pm 0.1}$	3.5$_{\pm 0.1}$	**16.5**$_{\pm 0.2}$	18.8$_{\pm 0.2}$
WRN-28-10 (200 epochs)	Cutout	**2.3**$_{\pm 0.1}$	2.6$_{\pm 0.1}$	**14.9**$_{\pm 0.2}$	16.9$_{\pm 0.1}$
WRN-28-10 (200 epochs)	AA	**2.1**$_{\pm <0.1}$	2.3$_{\pm 0.1}$	**13.6**$_{\pm 0.2}$	15.8$_{\pm 0.2}$
WRN-28-10 (1800 epochs)	Basic	**2.4**$_{\pm 0.1}$	3.5$_{\pm 0.1}$	**16.3**$_{\pm 0.2}$	19.1$_{\pm 0.1}$
WRN-28-10 (1800 epochs)	Cutout	**2.1**$_{\pm 0.1}$	2.7$_{\pm 0.1}$	**14.0**$_{\pm 0.1}$	17.4$_{\pm 0.1}$
WRN-28-10 (1800 epochs)	AA	**1.6**$_{\pm 0.1}$	2.2$_{\pm <0.1}$	**12.8**$_{\pm 0.2}$	16.1$_{\pm 0.2}$
Shake-Shake (26 2x96d)	Basic	**2.3**$_{\pm <0.1}$	2.7$_{\pm 0.1}$	**15.1**$_{\pm 0.1}$	17.0$_{\pm 0.1}$
Shake-Shake (26 2x96d)	Cutout	**2.0**$_{\pm <0.1}$	2.3$_{\pm 0.1}$	**14.2**$_{\pm 0.2}$	15.7$_{\pm 0.2}$
Shake-Shake (26 2x96d)	AA	**1.6**$_{\pm <0.1}$	1.9$_{\pm 0.1}$	**12.8**$_{\pm 0.1}$	14.1$_{\pm 0.2}$
PyramidNet	Basic	**2.7**$_{\pm 0.1}$	4.0$_{\pm 0.1}$	**14.6**$_{\pm 0.4}$	19.7$_{\pm 0.3}$
PyramidNet	Cutout	**1.9**$_{\pm 0.1}$	2.5$_{\pm 0.1}$	**12.6**$_{\pm 0.2}$	16.4$_{\pm 0.1}$
PyramidNet	AA	**1.6**$_{\pm 0.1}$	1.9$_{\pm 0.1}$	**11.6**$_{\pm 0.1}$	14.6$_{\pm 0.1}$
PyramidNet+ShakeDrop	Basic	**2.1**$_{\pm 0.1}$	2.5$_{\pm 0.1}$	**13.3**$_{\pm 0.2}$	14.5$_{\pm 0.1}$
PyramidNet+ShakeDrop	Cutout	**1.6**$_{\pm <0.1}$	1.9$_{\pm 0.1}$	**11.3**$_{\pm 0.1}$	11.8$_{\pm 0.2}$
PyramidNet+ShakeDrop	AA	**1.4**$_{\pm <0.1}$	1.6$_{\pm <0.1}$	**10.3**$_{\pm 0.1}$	10.6$_{\pm 0.1}$

Table 1: Results for SAM on state-of-the-art models on CIFAR-{10, 100} (WRN = WideResNet; AA = AutoAugment; SGD is the standard non-SAM procedure used to train these models).

Improves Performance across Models, Datasets, & Training Procedures

Model	Epoch	SAM Top-1	SAM Top-5	Standard Training (No SAM) Top-1	Standard Training (No SAM) Top-5
ResNet-50	100	**22.5**$_{\pm 0.1}$	6.28$_{\pm 0.08}$	22.9$_{\pm 0.1}$	6.62$_{\pm 0.11}$
	200	**21.4**$_{\pm 0.1}$	5.82$_{\pm 0.03}$	22.3$_{\pm 0.1}$	6.37$_{\pm 0.04}$
	400	**20.9**$_{\pm 0.1}$	5.51$_{\pm 0.03}$	22.3$_{\pm 0.1}$	6.40$_{\pm 0.06}$
ResNet-101	100	**20.2**$_{\pm 0.1}$	5.12$_{\pm 0.03}$	21.2$_{\pm 0.1}$	5.66$_{\pm 0.05}$
	200	**19.4**$_{\pm 0.1}$	4.76$_{\pm 0.03}$	20.9$_{\pm 0.1}$	5.66$_{\pm 0.04}$
	400	**19.0**$_{\pm <0.01}$	4.65$_{\pm 0.05}$	22.3$_{\pm 0.1}$	6.41$_{\pm 0.06}$
ResNet-152	100	**19.2**$_{\pm <0.01}$	4.69$_{\pm 0.04}$	20.4$_{\pm <0.0}$	5.39$_{\pm 0.06}$
	200	**18.5**$_{\pm 0.1}$	4.37$_{\pm 0.03}$	20.3$_{\pm 0.2}$	5.39$_{\pm 0.07}$
	400	**18.4**$_{\pm <0.01}$	4.35$_{\pm 0.04}$	20.9$_{\pm <0.0}$	5.84$_{\pm 0.07}$

Table 2: Test error rates for ResNets trained on ImageNet, with and without SAM.

Allows to increase # of epochs without overfitting

Improved Performance when Finetuning

Dataset	EffNet-b7 + SAM	EffNet-b7	Prev. SOTA (ImageNet only)	EffNet-L2 + SAM	EffNet-L2	Prev. SOTA
FGVC_Aircraft	$6.80_{\pm 0.06}$	$8.15_{\pm 0.08}$	**5.3** (TBMSL-Net)	**4.82**$_{\pm 0.08}$	$5.80_{\pm 0.1}$	5.3 (TBMSL-Net)
Flowers	**0.63**$_{\pm 0.02}$	$1.16_{\pm 0.05}$	0.7 (BiT-M)	**0.35**$_{\pm 0.01}$	$0.40_{\pm 0.02}$	0.37 (EffNet)
Oxford_IIIT_Pets	**3.97**$_{\pm 0.04}$	$4.24_{\pm 0.09}$	4.1 (Gpipe)	**2.90**$_{\pm 0.04}$	$3.08_{\pm 0.04}$	4.1 (Gpipe)
Stanford_Cars	$5.18_{\pm 0.02}$	$5.94_{\pm 0.06}$	**5.0** (TBMSL-Net)	$4.04_{\pm 0.03}$	$4.93_{\pm 0.04}$	**3.8** (DAT)
CIFAR-10	**0.88**$_{\pm 0.02}$	$0.95_{\pm 0.03}$	1 (Gpipe)	**0.30**$_{\pm 0.01}$	$0.34_{\pm 0.02}$	0.63 (BiT-L)
CIFAR-100	**7.44**$_{\pm 0.06}$	$7.68_{\pm 0.06}$	7.83 (BiT-M)	**3.92**$_{\pm 0.06}$	$4.07_{\pm 0.08}$	6.49 (BiT-L)
Birdsnap	**13.64**$_{\pm 0.15}$	$14.30_{\pm 0.18}$	15.7 (EffNet)	**9.93**$_{\pm 0.15}$	$10.31_{\pm 0.15}$	14.5 (DAT)
Food101	$7.02_{\pm 0.02}$	$7.17_{\pm 0.03}$	7.0 (Gpipe)	**3.82**$_{\pm 0.01}$	$3.97_{\pm 0.03}$	4.7 (DAT)
ImageNet	$15.14_{\pm 0.03}$	15.3	**14.2** (KDforAA)	**11.39**$_{\pm 0.02}$	11.8	11.45 (ViT)

Table 3: Top-1 error rates for finetuning EfficientNet-b7 (left; ImageNet pretraining only) and EfficientNet-L2 (right; pretraining on ImageNet plus additional data, such as JFT) on various downstream tasks. Previous state-of-the-art (SOTA) includes EfficientNet (EffNet) (Tan & Le, 2019),

Beats SOTA on several datasets

Robust to Label Noise

- Take X% of labels in train set and randomly flip them
- Beats other significantly more complicated ideas

Bootstrap = model is first trained as usual, then retrained from scratch on the labels predicted by the initially trained model

Method	Noise rate (%) 20	40	60	80
Sanchez et al. (2019)	94.0	92.8	90.3	74.1
Zhang & Sabuncu (2018)	89.7	87.6	82.7	67.9
Lee et al. (2019)	87.1	81.8	75.4	-
Chen et al. (2019)	89.7	-	-	52.3
Huang et al. (2019)	92.6	90.3	43.4	-
MentorNet (2017)	92.0	91.2	74.2	60.0
Mixup (2017)	94.0	91.5	86.8	76.9
MentorMix (2019)	**95.6**	**94.2**	91.3	**81.0**
SGD	84.8	68.8	48.2	26.2
Mixup	93.0	90.0	83.8	70.2
Bootstrap + Mixup	93.3	92.0	87.6	72.0
SAM	95.1	93.4	90.5	77.9
Bootstrap + SAM	95.4	**94.2**	**91.8**	79.9

Table 4: Test accuracy on the clean test set for models trained on CIFAR-10 with noisy labels. Lower block is our implementation, upper block gives scores from the literature, per Jiang et al. (2019).

Result Summary

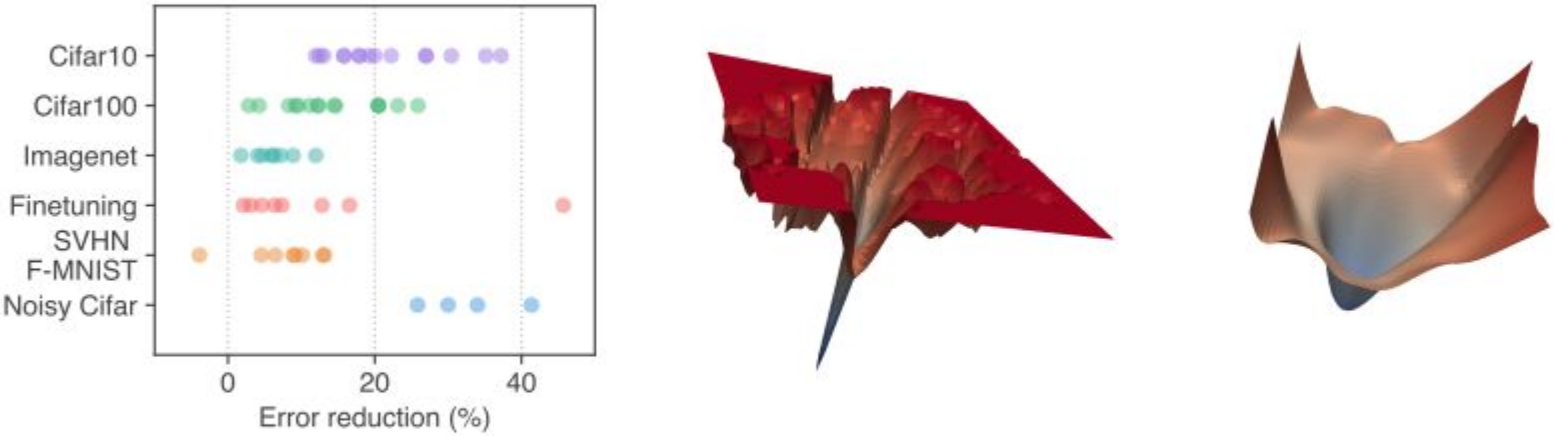
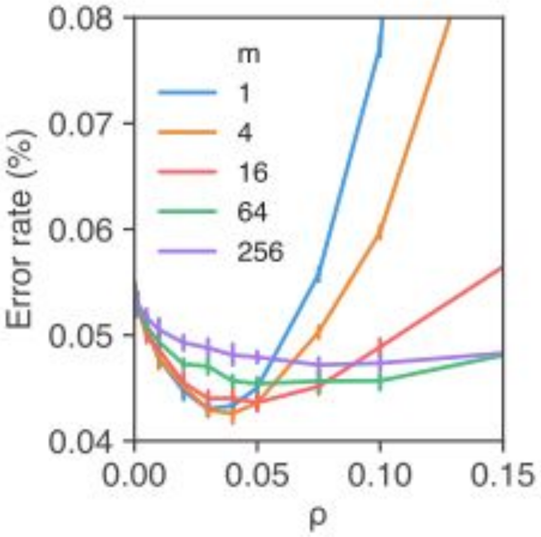


Figure 1: (left) Error rate reduction obtained by switching to SAM. Each point is a different dataset / model / data augmentation. (middle) A sharp minimum to which a ResNet trained with SGD converged. (right) A wide minimum to which the same ResNet trained with SAM converged.

m-sharpness

- Climb to local maxima (ϵ) is calculated with *m = per-batch / # gpus data points*
    - Smaller m generalizes better

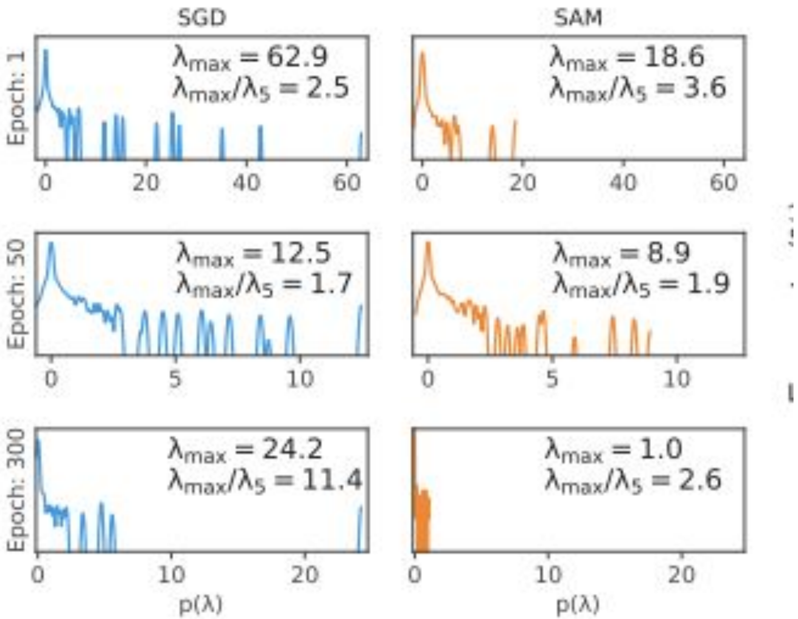


Kitware

Proof of Flat-ness

Evolution of the spectrum of the Hessian during training of a model with standard SGD (lefthand column) or SAM (righthand column)

Lower eigenvalues = flatter



Existing Code

TensorFlow/Jax: https://github.com/google-research/sam

PyTorch: https://github.com/davda54/sam

MosaicML: https://www.mosaicml.com/methods/sam

Usage

It should be straightforward to use SAM in your training pipeline. Just keep in mind that the training will run twice as slow, because SAM needs two forward-backward passes to estime the "sharpness-aware" gradient. If you're using gradient clipping, make sure to change only the magnitude of gradients, not their direction.

```python
from sam import SAM
...

model = YourModel()
base_optimizer = torch.optim.SGD  # define an optimizer for the "sharpness-aware" update
optimizer = SAM(model.parameters(), base_optimizer, lr=0.1, momentum=0.9)
...

for input, output in data:

  # first forward-backward pass
  loss = loss_function(output, model(input))  # use this loss for any training statistics
  loss.backward()
  optimizer.first_step(zero_grad=True)

  # second forward-backward pass
  loss_function(output, model(input)).backward()  # make sure to do a full forward pass
  optimizer.second_step(zero_grad=True)
...
```

Code

class **composer.algorithms.sam.SAM**(*rho: float = 0.05, epsilon: float = 1e-12, interval: int = 1*) [source]

Adds sharpness-aware minimization (Foret et al. 2020) by wrapping an existing optimizer with a `SAMOptimizer`.

Parameters:
- **rho** – The neighborhood size parameter of SAM. Must be greater than 0.
- **epsilon** – A small value added to the gradient norm for numerical stability.
- **interval** – SAM will run once per `interval` steps. A value of 1 will cause SAM to run every step. Steps on which SAM runs take roughly twice as much time to complete.

apply(*event: Event, state: State, logger: Optional[Logger]*)→ Optional[int] [source]

Applies SAM by wrapping the base optimizer with the SAM optimizer

Parameters:
- **event** (*Event*) – the current event
- **state** (*State*) – the current trainer state
- **logger** (*Logger*) – the training logger

match(*event: Event, state: State*)→ bool [source]

Run on Event.TRAINING_START

Parameters:
- **event** (`Event`) – The current event.

Overlap with Other Ideas

Learning Neural Network Subspaces

https://arxiv.org/pdf/2102.10472.pdf

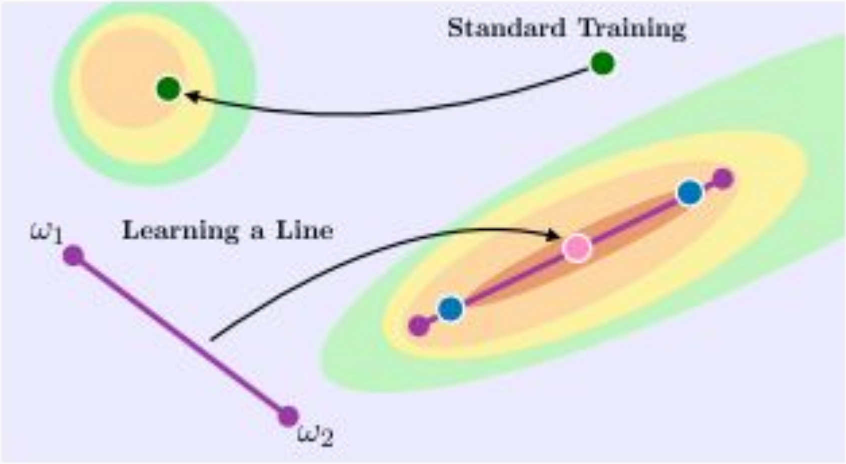


Figure 1. Schematic for learning a line of neural networks compared with standard training. The midpoint outperforms standard training in terms of accuracy, calibration, and robustness. Models near the endpoints enable high-accuracy ensembles in a single training run.