Sharpness-aware minimisation and label noise

Andrei-Tiberiu Alexandru

1 Introduction

Studying the relationship between flat minima and generalisation has yielded optimisation algorithms that are able to induce even higher accuracy for overparameterised networks, reaching state-of-the-art on high-profile benchmark tasks, such as image classification on the CIFAR-10 and CIFAR-100 datasets[FKMN20][Kri09]. While a universal characterisation of sharpness is elusive, the empirical backing motivates further exploration of algorithms that deliberately seek out flat minima. In this work, I investigate the behaviour of one such algorithm, sharpness-aware minimisation (SAM) [FKMN20], in the context of increasing levels of label noise. I choose to consider the noisy label regime because it has historically taught the research community useful lessons regarding how deep neural networks (DNNs) generalise[ZBH+21][AJB+17], a process that's critical for our understanding of deep learning. I also look at potential improvements to SAM, and formulate a set of experiments through which to evaluate them.

2 Related work

Recent literature connects features of the loss landscape with the capacity of neural networks to generalise. In particular, there is empirical support for the hypothesis that the generalisation gap (the difference in performance between training and test data) is larger for minima that are sharp, and smaller for those that are flat. Intuitively, a flat minimum is in a region where the training loss is more or less the same for the surrounding points in parameter space. There are several formal definitions of flatness [HS97], [KMN+16], though it is unclear which one most adequately characterises the phenomenon.

It is sometimes not meaningful to identify flat minima. For example, [DPBB17] show that for non-negative homogeneous networks there are infinite sets of parameters that are observationally equivalent - that is to say, they have the same input-to-output mapping. Within those sets, some parameters will correspond to flat minima and others to sharp minima – all without any change in the outputs. The derivations are done for deep rectified networks but also hold for convolutional networks. Moreover, for networks where the homogeneity property does not hold, unless a fixed parameterisation is considered, it isn't possible to claim a correlation between the flatness of the parameter space and the generalisation gap. This is because if a network can be reparameterised, it is possible to find an equivalent parameterisation where there are sharp minima at the exact points where there previously were flat minima, without changing the loss.

The practical implications of this correlation between flat minima and a smaller generalisation gap is that optimisation algorithms can be biased toward finding them, instead of simply minimising the loss function. There are several promising approaches such as entropy stochastic gradient descent[CCS+19], stochastic weight averaging (SWA)[IPG+18] and sharpness-aware minimisation (SAM)[FKMN20]. For example, SWA works by first applying stochastic gradient descent (SGD) for a fixed proportion of the total computational budget using a fixed learning rate to generate a pre-trained model $\hat{w}$. For the remaining budget, SGD is run with a cyclical learning rate, and the weights of iterates corresponding to minimum values of the learning rate are averaged together. The average of these weights defines a final model w^* which is used to make predictions. These algorithms differ in their training efficiency: entropy SGD and SWA seem to train in about the same time needed for plain SGD, sometimes faster, whereas SAM is notably slower.

Separately, there has been growing interest in supervised learning on noisy labels due to the potential to unlock much more data than is currently available. Datasets on which DNNs currently do well are carefully annotated, which can be time-consuming. Annotating a large dataset can be infeasible

if there are many classes, or where expert knowledge is required[RVBS17], for example in medical situations where a diagnosis is to be made based on X-ray scans. It would be useful if DNNs could generalise well from large datasets whose labels are only correct a fraction of the time. These datasets can be generated by crawling the web, for example, a situation in which it is impossible to go through every example and ensure it is correctly labelled.

This area of research is not new, with one of the most influential papers in recent years[ZBH+21] concluding that networks simply memorise the noise, and therefore train to convergence but generalise poorly. Since then, our view has become more nuanced, in that it seems that DNNs first learn patterns in the data, then memorise noise as training goes on[AJB+17].

Robustness to noise also depends heavily on the type of noise. In the setting in [ZBH+21], label noise is synthetic (also called "blue" noise in the rest of this work, following terminology from [JHLY20]), and it works as follows: replace the label of each example with another label in the possible classes with probability p. This means that the overall noise level of a dataset could be e.g. 20%, meaning only 80% of the examples have their true labels. Even within this subclass of noise, behaviour varies in a more complex way. In [RVBS17], the authors discover a critical threshold below which DNNs struggle to generalise: the label accuracy must be no lower than 1% above chance. For a dataset like CIFAR-10, where there are 10 possible classes, the probability of guessing correctly is one in 10, or 10%. Given this threshold, the label accuracy should be no lower than 11%, or the noise no higher than 89% of the dataset. [RVBS17] presents additional findings regarding the minimum size of the dataset that is necessary for generalisation given increasing noise, as well as how noise impacts the training batch size.

But datasets "in the wild" are not simply randomly labelled; where they are misclassified, their errors are systematic and skew in ways that aren't completely random. [WZC+21] introduces two companions to the original CIFAR-10 and CIFAR-100 datasets, in which the images are annotated by humans. These new datasets, CIFAR-10N and CIFAR-100N contain sets of labels comprising different levels of human noise, which [JHLY20] also calls "red" noise. A qualitative difference between red and blue noise is that red noise contains some relation to the image, visual or semantic [JHLY20]. For example, with blue noise, the label "orange" could be assigned to something completely unlike an orange, perhaps a ladybug or a wheel; with red noise, it would be assigned to something that is still in some way relevant: orange juice, or an orange bottle, or perhaps a photo of a person who was edited to seem more orange[WZC+21].

In a related line of research, several algorithms that are more robust to label noise have been produced. State of the art behaviour comes from dedicated algorithms such as MentorMix [JHLY20], Mixup [ZCDLP17] as well as from some that were not specifically designed with robustness in mind, such as sharpness-aware minimisation. In this work, I begin to explore further SAM's behaviour on noisy labels by replicating and extending the results in [FKMN20].

3 Sharpness-aware minimisation

Sharpness-aware minimisation (SAM) is an algorithm that optimises a modified loss function which takes into account some measure of the sharpness of the loss landscape [FKMN20]. The reasoning is that if sharpness is correlated with a higher generalisation gap, then we should be able to improve performance by simultaneously minimising both the error of the network and the sharpness of the surrounding region in weight space.

SAM uses a base optimiser such as stochastic gradient descent (SGD) to perform two conceptually straightforward tasks. The first is to find an adversarial ascent point $\mathbf{w_t} + \hat{\boldsymbol{\epsilon}}(\mathbf{w_t})$ that is in the neighbourhood of the current set of weights $\mathbf{w_t}$. Intuitively, this neighbourhood comprises points in parameter space that have similar values for training loss. Then, the gradient of the loss with respect to the adversarial point is computed, and the update is applied to the current set of weights $\mathbf{w_t}$, such that:

$$\mathbf{w}_{t+1} = \mathbf{w_t} - \eta \nabla L_{\mathcal{B}}(\mathbf{w})|_{\mathbf{w}+\hat{\boldsymbol{\epsilon}}(\mathbf{w})} \tag{1}$$

In other words, SAM finds the point in the neighbourhood of $\mathbf{w}_t$ which increases the loss the most. Then, it determines the direction which decreases the loss most from that point, and takes a step in that direction starting at the original weights $\mathbf{w}_t$. The size of the neighbourhood can be quantified as the p-norm of the difference of two weight vectors, and in SAM it is treated as a hyperparameter:

ρ. Finally, SAM also takes into account a regularisation term, which penalises large ℓ_2 norms of the weight vectors. The optimisation objective is:

$$\min_{\mathbf{w}} \max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} L_S(\mathbf{w} + \boldsymbol{\epsilon}) + \lambda \|\mathbf{w}\|_2^2 \tag{2}$$

In the experiments in [FKMN20], the norm of the perturbation $\boldsymbol{\epsilon}$ is generally fixed to be the ℓ_2 norm, but it can be generalised to any p-norm greater than 1 according to Appendix C.5 of the paper.

The performance of SAM is widely supported experimentally, offering state-of-the-art accuracy on several tasks including image classification [FKMN20]. As of the time of writing, when applied to EfficientNet-L2 [TL19], SAM holds the SOTA on image classification on CIFAR [pap]. However, one limitation of SAM is that it carries out back-propagation twice on each iteration: once to find the adversarial ascent point, with respect to the current weights $\mathbf{w}_t$ and another for the final gradient with respect to $\mathbf{w} + \boldsymbol{\epsilon}$. This makes it twice as slow as SGD, for example, which means that experiments comparing the two typically run SAM for half the number of epochs that SGD runs for [FKMN20]. This was observed experimentally and exacerbated even further by the bootstrap enhancement, see Section 5.

Another possible pitfall of using SAM is that some regularisation techniques may reduce the correlation between sharpness and generalisation gap by rescaling the parameters without impacting the loss. For example, weight decay reduces the norm of the weights, making it so that the initial neighbourhood size value is disproportionately large compared to the weights. This makes SAM overly choosy, seeking out a relatively large neighbourhood in which the loss landscape is flat. One way to address this is to apply a normalisation operator that essentially scales the size of the perturbation ϵ to negate the effect of the initial scaling operator[KKPC21]. The resulting implementation is called adaptive sharpness-aware minimisation (ASAM), and shows increased performance experimentally for the same computational cost.

Adapting the measure of sharpness to be in lock-step with the weights raises two questions. One is whether specific absolute values of ρ have any interpretation when it comes to the loss landscape, and the other is whether there is a way to update ρ during training such that "better" neighbourhoods are discovered. In the ideal case, minima would be part of large flat areas, but in practice this may not occur as often as we would like. Loss landscapes differ depending on the task, the architecture and more notably differ when noise is introduced.

4 Experiments

4.1 Experimental design

The experiments in this paper are largely following the design in §3.3 in [FKMN20] regarding robustness to label noise. I use a ResNet-32[HZRS16] architecture trained for 200 epochs using various levels of label noise and optimising algorithms. For some of the experiments I ran, hyperparameter choice slightly differs because of an oversight when I started running them (the exact hyperparameter details are in Appendix C of [FKMN20]. I did not go back to correct these because the experiments were relatively expensive to run given my computational budget (see Section 6 for a discussion). The differences are in learning rate (it is 0.1, should be 1.0) and weight decay (it is 0.0005 instead of 0.0001). Other parameters are identical: momentum is 0.9, label smoothing is 0.1 and batch size is 128. For the noise experiments, for SAM the neighbourhood size hyperparameter ρ is initially set to 0.1, whereas for ASAM it's set to 2.0.

The total number of experiments that I wanted to run is 320, broken down as follows[1]:

- 16 total label noise regimes comprising clean, 18% and 40% red noise, then 20%, 40%, 60%, 80% and 100% blue noise;
- 20 total combinations of optimiser, enhancement and neighbourhood size scheduler. These comprise SGD, SAM, ASAM and SWA, each with simple and bootstrapped counterparts. SAM and ASAM each have experiments for constant neighbourhood size during training as well as three neighbourhood size schedulers: step decay, exponential decay and step increase.

[1]For the full experimental setup please see: https://docs.google.com/spreadsheets/d/1k_A2HHLq378gvuAJ9gpHpc_a3X2uCbMY6H6pQ-WivMw/edit?usp=sharing

Optimiser	Number of epochs	Bootstrapped?	Duration	% slower than SGD
SGD	1	No	9 minutes, 30 seconds	0%
SGD	1	Yes	22 minutes, 8 seconds	132%
SAM	1	No	17 minutes, 41 seconds	86%
SAM	1	Yes	40 minutes, 2 seconds	321%

Table 1: The duration of one training epoch on a single CPU.

4.2 Noise datasets

The datasets I used were CIFAR-10 and CIFAR-100[WZC+21]. Synthetic noise is equivalent to flipping the label of an image to a random label with probability p. In the experiments I use 0.2, 0.4, 0.6, 0.8 and 1.0, similarly to [FKMN20]. "Red" or natural noise results from misclassifications by humans of the images in CIFAR, and was collected by [WZC+21] into the datasets CIFAR-10N and CIFAR-100N. We use the two highest-noise labels, "aggregate" (18% noise) and "worst" (40% noise).

4.3 Implementation

The experiments were run on an existing open-source PyTorch[PGM+19] implementation of SAM[Sam21], and used label randomisation code from a replication of [ZBH+21] by one of its authors [Zha17]. The implementation of ResNet-32[HZRS16] is due to [Ide]. I built a wrapper around the SAM training procedure to allow more flexible experiments to be run from the command line and added functionality to load in the red noise labels. Experiments were scripted to run unsupervised from a remote terminal.

I also implemented bootstrapping on top of the SGD and SAM training procedures to compare with [JHLY20]. Bootstrapping is a simple procedure where a given model is first trained on the data, then trained again using its own predictions on the training set as ground truth labels. The idea is that training on the predictions acts as a resampling of the random variables that generated the labels in the first place, and allows the model to better learn the underlying probability distribution of the data. The downside of bootstrapping – for neural networks at least – is that it requires double the amount of computation, which can be prohibitively expensive depending on the architecture, for only marginal gains in performance.

To investigate the behaviour of the neighbourhood size hyperparameter I adapted the learning rate schedule used in the SAM implementation to generate three schedules for ρ. The first is an epoch-wise decay of the neighbourhood size, where:

- for the first 30% of the training run, ρ retains its initial value
- between 30%-60% of the run, it is reduced by 90% from the initial value;
- for the rest of the run it is reduced a further 90%;

The second is the exponential decay schedule, which implements:

$$\rho_t = \rho_0 \cdot \exp(-kt) \tag{3}$$

where k is a hyperparameter that defaults to 0.1. Finally, there is a simple step-increase schedule that increases the neighbourhood size by a factor of 10 in a similar fashion to the step decay described above.

4.4 Hardware

The experiments were run on a cloud setup running Ubuntu 20.04 with 40 AMD Epyc Rome vCPUs, 4 nVIDIA A100 for PCIe GPUs with 40Gb of VRAM each, 256Gi of RAM and 40Gi of storage. I also ran some test experiments locally to measure time taken by the different optimiser combinations. For details on how compute restrictions have affected these experiments, see Section 6.

Optimiser	Train/test	CIFAR-10 clean	CIFAR-10 18% red noise	CIFAR-10 40% red noise	CIFAR-10 20% blue noise	CIFAR-10 80% blue noise	CIFAR-10 100% blue noise
SGD	Train accuracy	100%	99.98%	100%	100%	100%	10.29%
	Test accuracy	87.29%	81.67%	55.11%	70.90%	18.32%	10.00%

Table 2: Results of experiments on SGD.

5 Preliminary results

The results obtained so far are unfortunately only a subset of the experiments I would want to carry out, and do not span the most interesting use cases. I started with SGD on all noise combinations so that I would have a baseline against which to compare SAM and ASAM. I earmarked the 40% and 60% blue noise cases for later as they were less extreme than the 0%, 20%, 80% and 100% cases, but in retrospect it would've been useful to run the 40% case to compare with the 40% red noise. The test accuracy gradually decreases as the noise is increased, with the accuracy at 100% noise corresponding to the 1 in 10 chance of being correct when selecting a random label on CIFAR-10. Interestingly, in the 100% blue noise case, the network struggles to converge. Looking at the full log[2], most of the training seems wasted, in that the highest test accuracy of 11.42% is already reached in epoch 8/200. Two runs that are almost comparable are the 18% red noise and 20% blue noise settings. To reiterate, red noise is noise that results from human misclassification of the images, whereas blue noise is a simple procedure by which a label gets flipped to another random label with probability p before training. Intuitively, this results in different types of noise, because humans are quite unlikely to label an image of a man holding a fish as an aeroplane, but they are likely to label the image as fish, when man is correct, or vice-versa. It seems that in this setting, the converged network is over 10% more accurate on red noise than on blue noise, indicating that red noise indeed still retains some connection to the information in the image, whereas blue noise is completely random.

6 Discussion

In this paper I begin to investigate further the behaviour of SAM on image classification with noisy labels. In [FKMN20], SAM shows good robustness to synthetic noise. Here, I try to look at whether there are any differences in performance between red and blue noise, as well as at the impact of different types scheduler for the neighbourhood size. I did not manage to run all the experiments because of the computational cost of training. The setup detailed in Section 4.4 cost around $10 per hour from a dedicated cloud provider, and with each experiment taking about 8s per epoch or 27 minutes in total, the overall cost would have been $1,600.

The authors of [FKMN20] explore applications of SAM to different architectures, but these skew toward the overparameterised regime where it's likely that SAM would have a higher impact. As a result, we don't know how SAM interacts with smaller networks. For example, I ran some preliminary experiments on a ∼60,000-parameter convolutional network (in the context of CIFAR-10, this should be just about overparameterised) and did not find a marked improvement of SAM over SGD, although it didn't seem like my network was able to converge regardless. Perhaps more training was needed. Again, the additional cost of SAM in relation to SGD is non-trivial without additional measures to parallelise training.

7 Further work

This paper does not report conclusive findings due to computational limitations. It was designed as an open-ended investigation of the behaviour of SAM on noisy labels that started out as an effort to replicate the findings in the section on robustness to noise in [FKMN20]. The obvious next step would be to carry out all the experiments, or at least a subset of them that reveal some of the behaviour of

[2]Logs and code: https://github.com/inwaves/sam-noisy-labels

SAM when using heavily corrupted labels. It is possible to speed up the experiments by implementing data parallelism between GPU accelerators, as briefly mentioned in §3.1 of the original SAM paper. I tried to get this to work on my cloud setup, but as it was taking a lot of time I decided to deprioritise it.

Another interesting avenue of research would be comparing SAM against stochastic weight averaging (SWA)[IPG+18], another optimiser that tends to favour flat minima. SWA is now natively implemented in Pytorch, making it straightforward to use. It is also potentially faster to run, taking around the same time as simple SGD. The authors themselves carry out a preliminary analysis in response to a review[ope], however the results did not make it to the main paper.

I am most excited about the behaviour of the neighbourhood size hyperparameter in practice. Analysing more closely what sort of values work and which harm performance could serve to show what sort of "neighbourhoods" each type of task has, and perhaps where they tend to be. Two things were not explored in the original paper. First, changing the neighbourhood size during training for the purpose of honing in on a suitable neighbourhood could be useful. This is similar to learning rate schedules, where the learning rate decays as training goes on. It could be the case that always seeking the same type of neighbourhood works best. It's also conceivable that during training, as it becomes obvious that large flat regions do not exist, an optimiser could do better if it was "satisfied" with a smaller flat neighbourhood. Second, it would be interesting to perform hyperparameter optimisation of ρ in a more concerted way. In the original paper ρ is chosen through grid search over a set of 6 possible values equally spaced on a log scale. Given enough computational resources, Bayesian optimisation using a Gaussian process could yield better results.

References

[AJB+17] Devansh Arpit, Stanisław Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *International Conference on Machine Learning*, pages 233–242. PMLR, 2017.

[CCS+19] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019.

[DPBB17] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pages 1019–1028. PMLR, 2017.

[FKMN20] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural computation*, 9(1):1–42, 1997.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[Ide] Yerlan Idelbayev. Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch. https://github.com/akamaster/pytorch_resnet_cifar10. Accessed: 2022-01-19.

[IPG+18] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[JHLY20] Lu Jiang, Di Huang, Mason Liu, and Weilong Yang. Beyond synthetic noise: Deep learning on controlled noisy labels. In *International Conference on Machine Learning*, pages 4804–4815. PMLR, 2020.

[KKPC21] Jungmin Kwon, Jeongseop Kim, Hyunseo Park, and In Kwon Choi. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. *arXiv preprint arXiv:2102.11600*, 2021.

[KMN+16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[ope] Sharpness-aware minimization - iclr2021 open review. https://openreview.net/forum?id=6Tm1mposlrM. Accessed: 2022-01-19.

[pap] Papers with code - cifar-100 benchmark (image classification). https://paperswithcode.com/sota/image-classification-on-cifar-100. Accessed: 2022-01-19.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[RVBS17] David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017.

[Sam21] David Samuel. Sharpness-aware minimization (pytorch). https://github.com/davda54/sam, 2021.

[TL19] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[WZC+21] Jiaheng Wei, Zhaowei Zhu, Hao Cheng, Tongliang Liu, Gang Niu, and Yang Liu. Learning with noisy labels revisited: A study using real-world human annotations. *arXiv preprint arXiv:2110.12088*, 2021.

[ZBH+21] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[ZCDLP17] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[Zha17] Chiyuan Zhang. Fitting random labels. https://github.com/pluskid/fitting-random-labels/, 2017.