Large batch sizes for Deep Learning

Mario Rene Surlemont
University Vienna

Elnaz Javadi Farahzadi
University Vienna

Alexander Stähle
University Vienna

***Abstract*—A critical decision when using SGD variants is choosing the size of a batch. In the past, it has been shown that a generalization gap occurs when using large batch sizes rather than small batch sizes. This can be compensated (conditionally) by a larger learning rate, but the minima found usually remain sharper than with smaller batch sizes. Within this project we evaluate whether a formulation that regulates the sharpness of minimizers is suitable to compensate for the problems of large batch sizes. We do this by assessing the results of an empirical study of a heterogeneous set of optimizers and loss functions in relation to different batch sizes.**

I. Introduction

Deep Learning has become a popular approach to solve (among others) classification problems. The non-convex optimization problem to be solved in Deep Learning applications is typically of the form

$$\min_{w\in\mathbb{R}^d} f(w) := \frac{1}{M}\sum_{i=1}^{M} f_i(w), 1 \tag{1}$$

where f_i is the loss function that calculates the loss for data point $i \in \{1, 2, ..., M\}$, M is the number of data points and w is the weight vector. For minimizing such functions one often uses Stochastic Gradient Descent (SGD) and its variants. f is minimized by iteratively taking steps of the form

$$x_{k+1} = x_k - \alpha_k\left(\frac{1}{|B_k|}\sum_{i=1}^{|B_k|}\nabla f_i(w_k)\right). \tag{2}$$

Here $B_k \subset \{1, 2, ..., M\}$ is a randomly selected batch of size $|B_k|$ and α_k is the learning rate in iteration step k. Usually one uses batches of size $|B_k| \in \{32, 64, ..., 512\}$, where $|B_k| \ll M$.

Since the calculation of the gradient of the loss function cannot be parallelised well when using these rather small batches and can therefore increase the runtime considerably, the influence of large batch sizes on the learning process and the final result has been investigated in the past.

In general, it appears that using larger batches results in a larger generalization gap with one possible reasoning behind this being so-called "sharp minima" [1]. This can be partially compensated by an adjusted learning rate [2], but the sharp minima remain. Therefore, efforts have been made to develop a problem formulation that not only minimizes the given loss function, but simultaneously adjusts the sharpness of the minima [3]. This formulation is referred to as Sharpness-Aware Minimization (SAM).

Within this project, we will now evaluate to what extent SAM is suitable to enable training with large batches.

We will train a very simple neural network for image recognition once using SGD with small to moderate batch sizes and apply SAM to large batches for comparison. Furthermore we will explore how choosing different variants of SGD (e.g. Adam) will affect the generalization ability of the model.

The resulting minima will additionally be examined for sharpness and runtime. We also evaluate how learning rate and batch size are related in the SGD scenario and check whether this relationship is also evident when SAM is used.

In [1]

In Section 2 we will look into why training with large batch sizes is not as popular in deep learning applications. After that, Section 3 will cover SAM and the idea behind it. Section 4 will cover the experiments we conducted and we conclude this work with a short discussion.

II. Drawbacks of large batch sizes

With SGD being one of the most prominent algorithms, it is used as a default method in numerous applications. While there are many different parameters to be tuned and perfected choosing a batch size is one of the most important ones to optimize. It is often observed that smaller batch sizes lead to a better generalization, while larger batch sizes seem to degrade the quality of the model. In this section, we will look into the known causes that are the fundamental issues we want to explore in this work.

A. Known issues

While small batch sizes, in theory, are not guaranteed to converge to a global minimum, they seem to outperform large batch sizes repeatedly when looking at the error on the test set [1]. Interestingly, the error on the training set seems to be similar to that of small batch sizes. This discrepancy between train and test error is also referred to as the generalization gap. Practitioners seem to run into this generalization gap, especially in large batch learning [5]. Still large batch sizes allow for parallelized computations in a manner that is way above the possibilities of parallelizing small batch training even considering efforts to overcome this shortcoming (e.g. [6]). Parallelization leads to huge speed-ups but as long as large batch sizes suffer larger generalization gaps this time-saving property does not outweigh the advantages of smaller batches.

B. Our Observations

In order for our following experiments to have a foundation we conducted similar studies as in previous works (see [7], [8]) which were trying to look into causes of the larger

generalization gap. Our observations are based on a simple deep learning model consisting of two convolutional layers with 32 and 64 filters, ReLU activations and intermediate max-pooling and drop-out layers and a single dense layer at the end. Even on this small model we could observe higher generalization gaps for larger batch sizes which reached up to $XX\%$ (TODO: fill in a number here) (see Appendix).

C. Sharpness of Minima

Keskar et al make the interesting observation that models trained with large batch sizes seem to tend to converge to sharp minima rather than flat minima which is the case for small batch sizes [1]. This is a possible explanation for the above-mentioned generalization gap. The model might perform well on the training data but as the test data is generally not identical to the training data even small shifts in the optimal minima from training set to test set can lead to significant differences in the error. Small batch sizes however converge to flatter minima which is more robust to any shifts of the minimum, leading to a better performance on the test set. It is therefore assumed to be desirable for a model to converge to flatter minima.

To encourage flatter minima Keskar et al attempt a number of techniques involving data augmentation and conservative training but while they do improve the generalization of models with large batch sizes the sharpness of the minima stays the same [1]. We therefore turn to a work of Foret et al in which they present an easy-to-use and effective way to penalize sharp minima [3]. In the following sections we will explain the idea behind their approach and evaluate if this idea remedies the problem of large batches.

III. SAM

In [3] a problem formulation was presented in which an arbitrary loss function of the form (**??**) can be minimized simultaneously with its sharpness. The basic goal is to ensure that the value of the loss function does not change too much within a ϵ-environment of the p-norm.

The basic problem formulation here is

$$\min_{\omega} L_S^{SAM}(\omega) + \lambda ||\omega||_2^2, \tag{3}$$

where

$$L_S^{SAM} := \max_{||\epsilon||_p \leq \rho} L_S(\omega + \epsilon). \tag{4}$$

ρ is a hyperparameter and p indicates which norm is used. The authors show that in general $p = 2$ is the optimal choice. Therefore, we also use this parameter in the further course.

In order to minimize this problem, the authors propose an approximation of the gradient

$$\nabla_\omega L_S^{SAM}(\omega) + \lambda ||\omega||_2^2 \approx \nabla_\omega L_S(\omega)_{|\omega + \hat{\epsilon}(\omega)}, \tag{5}$$

where

$$\hat{\epsilon}(\omega) := \arg\max_{||\epsilon||_p \leq \rho} L_S(\omega + \epsilon) = \rho \frac{\nabla_\omega L_S(\omega)}{|\nabla_\omega L_S(\omega)||_2}, \tag{6}$$

which can be computed by standard libraries like TensorFlow via autodifferentiation. This results in an problem formulation that can be minimized using SGD or ADAM.

The resulting SAM algorithm is as follows:

Algorithm 1 SAM-Algorithmus

Require: $X := (\cup_{i=1}^{n} \{x_i, y_i\})$, Loss function l, Batch size b, Optimizer o, Neighborhood size ρ
Ensure: Model M_S trained with SAM
  Initialize weights $\omega_0, t = 0$;
  while $not\, converged$ **do**
    $\mathcal{B} \leftarrow \{(x_1, y_1), ..., (x_b, y_b)\}$; ▷ a randomly chosen batch of size b
    Compute gradient $L_\omega L_\mathcal{B}(\omega)$ of the batch's training loss; ▷ Step 1
    Compute $\hat{\epsilon}(\omega)$ via (**??**); ▷ Step 2
    Compute $g := \nabla_\omega L_\mathcal{B}(\omega)_{|\omega + \hat{\epsilon}(\omega)}$; ▷ Step 3
    Update weights ω_{t+1} via optimizer o using g;
    $t \leftarrow t + 1$
  end while

IV. OLD STUFF

The aim of writing a paper is to infect the mind of your reader with the brilliance of your idea [**?**]. The hope is that after reading your paper, the audience will be convinced to try out your idea. In other words, it is the medium to transport the idea from your head to your reader's head. In the following section, we show a common structure of scientific papers and briefly outline some tips for writing good papers in Section V.

At that point, it is important that the reader is able to reproduce your work [**?**], [**?**], [**?**]. This is why it is also important that if the work has a computational component, the software associated with producing the results are also made available in a useful form. Several guidelines for making your user's experience with your software as painless as possible is given in Section VI.

This brief guide is by no means sufficient, on its own, to make its reader an accomplished writer. The reader is urged to use the references to further improve his or her writing skills.

V. The Structure of a Paper

Scientific papers usually begin with the description of the problem, justifying why the problem is interesting. Most importantly, it argues that the problem is still unsolved, or that the current solutions are unsatisfactory. This leads to the main gist of the paper, which is "the idea". The authors then show evidence, using derivations or experiments, that the idea works. Since science does not occur in a vacuum, a proper comparison to the current state of the art is often part of the results. Following these ideas, papers usually have the following structure:

Abstract

Short description of the whole paper, to help the reader decide whether to read it.

Introduction
: Describe your problem and state your contributions.

Models and Methods
: Describe your idea and how it was implemented to solve the problem. Survey the related work, giving credit where credit is due.

Results
: Show evidence to support your claims made in the introduction.

Discussion
: Discuss the strengths and weaknesses of your approach, based on the results. Point out the implications of your novel idea on the application concerned.

Summary
: Summarize your contributions in light of the new results.

VI. Tips for Good Writing

The ideas for good writing have come from [**?**], [**?**], [**?**].

A. Getting Help

One should try to get a draft read by as many friendly people as possible. And remember to treat your test readers with respect. If they are unable to understand something in your paper, then it is highly likely that your reviewers will not understand it either. Therefore, do not be defensive about the criticisms you get, but use it as an opportunity to improve the paper. Before your submit your friends to the pain of reading your draft, please *use a spell checker*.

B. Abstract

The abstract should really be written last, along with the title of the paper. The four points that should be covered [**?**]:

1) State the problem.
2) Say why it is an interesting problem.
3) Say what your solution achieves.
4) Say what follows from your solution.

C. Figures and Tables

Use examples and illustrations to clarify ideas and results. For example, by comparing Figure 1 and Figure 2, we can see the two different situations where Fourier and wavelet basis perform well.

D. Models and Methods

The models and methods section should describe what was done to answer the research question, describe how it was done, justify the experimental design, and explain how the results were analyzed.

The model refers to the underlying mathematical model or structure which you use to describe your problem, or that your solution is based on. The methods on the other hand, are the algorithms used to solve the problem. In some cases, the suggested method directly solves the problem, without having it stated in terms of an underlying model. Generally though it is a better practice to have the model figured out and stated clearly, rather than presenting a method without specifying the model. In this case, the method can be more easily evaluated in the task of fitting the given data to the underlying model.

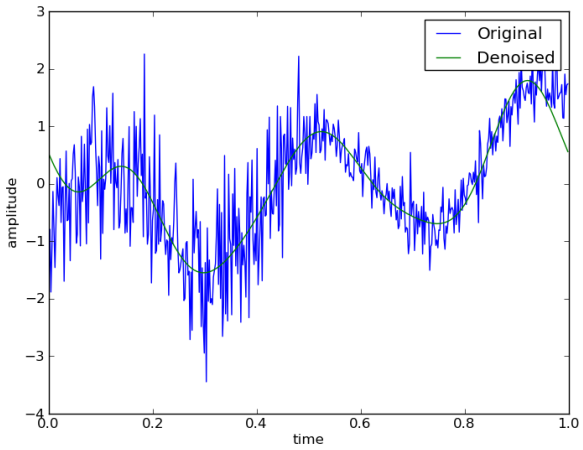

Fig. 1. Signal compression and denoising using the Fourier basis.

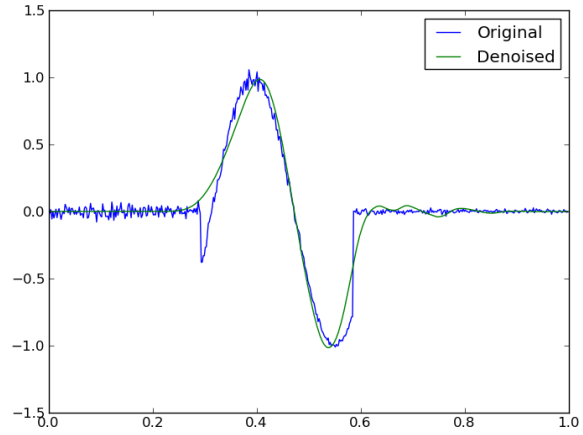

Fig. 2. Signal compression and denoising using the Daubechies wavelet basis.

The methods part of this section, is not a step-by-step, directive, protocol as you might see in your lab manual, but detailed enough such that an interested reader can reproduce your work [**?**], [**?**].

The methods section of a research paper provides the information by which a study's validity is judged. Therefore, it requires a clear and precise description of how an experiment was done, and the rationale for why specific experimental procedures were chosen. It is usually helpful to structure the methods section by [**?**]:

1) Layout the model you used to describe the problem or the solution.
2) Describing the algorithms used in the study, briefly including details such as hyperparameter values (e.g.

thresholds), and preprocessing steps (e.g. normalizing the data to have mean value of zero).
3) Explaining how the materials were prepared, for example the images used and their resolution.
4) Describing the research protocol, for example which examples were used for estimating the parameters (training) and which were used for computing performance.
5) Explaining how measurements were made and what calculations were performed. Do not reproduce the full source code in the paper, but explain the key steps.

E. Results

Organize the results section based on the sequence of table and figures you include. Prepare the tables and figures as soon as all the data are analyzed and arrange them in the sequence that best presents your findings in a logical way. A good strategy is to note, on a draft of each table or figure, the one or two key results you want to address in the text portion of the results. The information from the figures is summarized in Table I.

When reporting computational or measurement results, always report the mean (average value) along with a measure of variability (standard deviation(s) or standard error of the mean).

VII. Tips for Good Software

There is a lot of literature (for example [**?**] and [**?**]) on how to write software. It is not the intention of this section to replace software engineering courses. However, in the interests of reproducible research [**?**], there are a few guidelines to make your reader happy:

- Have a `README` file that (at least) describes what your software does, and which commands to run to obtain results. Also mention anything special that needs to be set up, such as toolboxes[1].
- A list of authors and contributors can be included in a file called `AUTHORS`, acknowledging any help that you may have obtained. For small projects, this information is often also included in the `README`.
- Use meaningful filenames, and not `temp1.py`, `temp2.py`.
- Document your code. Each file should at least have a short description about its reason for existence. Non obvious steps in the code should be commented. Functions arguments and return values should be described.
- Describe how the results presented in your paper can be reproduced.

A. LaTeX Primer

LaTeX is one of the most commonly used document preparation systems for scientific journals and conferences. It is based on the idea that authors should be able to focus on the content of what they are writing without being distracted by its visual presentation. The source of this file can be used as a starting point for how to use the different commands in LaTeX. We are using an IEEE style for this course.

1) Installation: There are various different packages available for processing LaTeX documents. See our webpage for more links for getting started.

2) Compiling LaTeX: Your directory should contain at least 4 files, in addition to image files. Images should ideally be `.pdf` format (or `.png`).

3) Equations: There are three types of equations available: inline equations, for example $y = mx + c$, which appear in the text, unnumbered equations

$$y = mx + c,$$

which are presented on a line on its own, and numbered equations

$$y = mx + c \tag{7}$$

which you can refer to at a later point (Equation (3)).

4) Tables and Figures: Tables and figures are "floating" objects, which means that the text can flow around it. Note that `figure*` and `table*` cause the corresponding figure or table to span both columns.

VIII. Summary

The aim of a scientific paper is to convey the idea or discovery of the researcher to the minds of the readers. The associated software package provides the relevant details, which are often only briefly explained in the paper, such that the research can be reproduced. To write good papers, identify your key idea, make your contributions explicit, and use examples and illustrations to describe the problems and solutions.

Appendix

Additional conducted experiments: TODO put plots of initial vanilla generalization gap (val loss / train loss)

[1]For those who are particularly interested, other common structures can be found at http://en.wikipedia.org/wiki/README and http://www.gnu.org/software/womb/gnits/.

Basis	Support	Suitable signals	Unsuitable signals
Fourier	global	sine like	localized
wavelet	local	localized	sine like

TABLE I
CHARACTERISTICS OF FOURIER AND WAVELET BASIS.

REFERENCES

[1] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," 2017.

[2] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2018.

[3] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," 2021.

[4] J. Kwon, J. Kim, H. Park, and I. K. Choi, "ASAM: Adaptive Sharpness-Aware Minimization for Scale-Invariant Learning of Deep Neural Networks," in *Proceedings of the 38th International Conference on Machine Learning*. PMLR, Jul. 2021, pp. 5905–5914.

[5] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade: Second Edition*, ser. Lecture Notes in Computer Science, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer, 2012, pp. 9–48.

[6] D. Das, S. Avancha, D. Mudigere, K. Vaidynathan, S. Sridharan, D. Kalamkar, B. Kaul, and P. Dubey, "Distributed Deep Learning Using Synchronous Stochastic Gradient Descent," *arXiv:1602.06709 [cs]*, Feb. 2016.

[7] D. Chang, "Effect of Batch Size on Neural Net Training," Aug. 2020.

[8] K. Shen, "Effect of batch size on training dynamics," Jun. 2018.