OPT2021: 13th Annual Workshop on Optimization for Machine Learning

Adaptive Optimization with Examplewise Gradients

Julius Kunze JULIUSKUNZE@GMAIL.COM
James Townsend JAMES.TOWNSEND@CS.UCL.AC.UK
David Barber DAVID.BARBER@CS.UCL.AC.UK
Department of Computer Science
University College London

Abstract

We propose a new, more general approach to the design of stochastic gradient-based optimization methods for machine learning. In this new framework, optimizers assume access to a *batch* of gradient estimates per iteration, rather than a single estimate. This better reflects the information that is actually available in typical machine learning setups. To demonstrate the usefulness of this generalized approach, we develop Eve, an adaptation of the Adam optimizer which uses examplewise gradients to obtain more accurate second-moment estimates. We provide preliminary experiments, without hyperparameter tuning, which show that the new optimizer slightly outperforms Adam on a small scale benchmark and performs the same or worse on larger scale benchmarks. Further work is needed to refine the algorithm and tune hyperparameters.

1. Introduction

In many practical settings in machine learning, we would like to solve a problem of the form

$$\arg\min_{\theta} \mathbb{E}[f(x, \theta)]. \tag{1}$$

We do not know the underlying distribution of x, but we do have access to a (often large) set of samples $X = (x_1, \ldots, x_D)$ from the distribution. In this situation, we optimize the approximate loss function

$$L(\theta) := \frac{1}{D} \sum_d f(x_d, \theta) \approx \mathbb{E}[f(x, \theta)], \tag{2}$$

using stochastic gradient estimates of the form

$$\overline{g}(\theta) := \frac{1}{B} \sum_{b=1}^{B} \nabla_\theta f(x_{d_b}, \theta), \tag{3}$$

where $d_1, \ldots, d_B$ is a random sub-sample of the index set $\{1, \ldots, D\}$, drawn at each iteration of an optimization process. We refer to B as the 'batch size'.

Optimization methods based on stochastic gradients typically assume access to the estimator $\overline{g}$, without making use of the 'examplewise' gradient estimates $\nabla_\theta f(x_{d_b}, \theta)$. For optimizers whose iterates depend only linearly on $\overline{g}$, this makes sense—the empirical mean over the batch is then no more informative than the examplewise estimates. However, state-of-the-art methods, such as the Adam optimizer [9], typically have nonlinear dependence on $\overline{g}$, and in these situations it may be beneficial to make direct use of the examplewise estimates. Especially at the beginning of training, Adam's second-moment estimates can be unstable, which can harm learning [12].

Our contribution is twofold. Firstly, we propose *Eve*, an adaptation of Adam that uses a more accurate *examplewise variance estimate* of the gradient. Our preliminary experiments show it can lead to improved performance over Adam in some cases.

These results for the Eve algorithm suggest a more general insight, that when designing gradient-based optimizers for machine learning, one should assume access not to a single gradient estimate per iteration, but to a batch of i.i.d. gradient estimates. This better reflects the information that is actually available, given the common practice of evaluating functions on batches of data in parallel, on modern hardware such as graphics processing units (GPUs).

Based on this idea, we propose a new framework for designing gradient-based optimization algorithms. We show that arbitrary statistics of examplewise gradients for adaptive optimization, including higher-order moments, the median and mean absolute deviation, can be obtained using standard software tools. In contrast, previous work such as BackPACK [4] relies on a specialized algorithm [3, 5] that can efficiently obtain statistics derived from the examplewise second moment. We provide a model-agnostic example implementation that supports any differentiable network architecture, including residual [7], recurrent [8] and attention-based [18] networks. This is in contrast to BackPACK, which requires custom code per operation type and only supports chains of linear layers with activations, without fan-in or fan-out.

On a perfectly parallel machine, computing other statistics in addition to the mean should incur minimal per-iteration runtime overhead for deep enough models. We find that in practice runtime overhead varies depending on the architecture. We demonstrate the feasibility of our more general approach by testing Eve on large-scale problems but leave wider exploration of this new optimizer design space for future work.

2. Examplewise Variance Estimation (Eve)

In this section, we motivate and derive Eve, a first-order optimizer based on Adam [9], making use of examplewise gradients. For non-batched optimization ($B = 1$), Eve is equivalent to Adam, inheriting its convergence guarantees. While not evaluated in this paper, it is straightforward to transfer the idea of Eve to other adaptive optimizers based on batch gradient second moments, such as Adadelta [19] and Rmsprop [6].

Adam computes exponential moving average statistics $\widehat{m}$ and $\widehat{v}$ for the first and second moment $\mathbb{E}\overline{g}$ and $\mathbb{E}\overline{g}^2$ of the batch mean gradient $\overline{g}$. At each timestep, parameters are updated via

$$\theta \leftarrow \theta - \alpha \frac{\widehat{m}}{\sqrt{\widehat{v}} + \epsilon}, \tag{4}$$

where ϵ is a hyperparameter which is usually small and intended to improve numerical stability. Ignoring ϵ, the update can be rewritten as

$$\theta \leftarrow \theta - \operatorname{sign}(\widehat{m}) \frac{\alpha}{\sqrt{\overline{\eta}^2 + 1}}, \tag{5}$$

where $\overline{\eta} := \sqrt{\frac{\widehat{v} - \widehat{m}^2}{\widehat{m}^2}}$ is a statistic for the batch gradient's relative standard deviation $\frac{\sqrt{\mathrm{Var}(\overline{g})}}{|\mathbb{E}\overline{g}|}$ [2].

We therefore propose to use an exponential moving average statistic $\widehat{v}_e$ of the second moment $\mathbb{E}g^2$ of the examplewise gradient g to obtain a more accurate batch gradient variance estimate. Due to the variance sum law, batch gradient variance $\text{Var}(\overline{g})$ is related to the examplewise variance $\text{Var}(g)$ via $\text{Var}(\overline{g}) = \frac{1}{B}\text{Var}(g)$, resulting in

$$\frac{\sqrt{\text{Var}(\overline{g})}}{|\mathbb{E}\overline{g}|} = \frac{\sqrt{\frac{1}{B}\text{Var}(g)}}{|\mathbb{E}\overline{g}|}. \tag{6}$$

This motivates the use of $\eta := \sqrt{\frac{1}{B}\frac{\widehat{v}_e - \widehat{m}^2}{\widehat{m}^2}}$ instead of $\overline{\eta}$ to estimate the relative batch gradient standard deviation. For a normally distributed, slowly-changing gradient distribution, the standard deviation of the estimator η ('standard error') is approximately $\sqrt{B}$ lower than $\overline{\eta}$'s (derivation in appendix A), demonstrating its improved stability. Using η results in the following update for Eve:

$$\theta \leftarrow \theta - \alpha \frac{\widehat{m}}{\sqrt{\frac{1}{B}\widehat{v}_e + \left(1 - \frac{1}{B}\right)\widehat{m}^2} + \epsilon}. \tag{7}$$

The expression $\frac{1}{B}\widehat{v}_e + \left(1 - \frac{1}{B}\right)\widehat{m}^2$ is a second moment statistic of the batch gradient $\overline{g}$. Algorithm 1 shows the full pseudo-code for the Eve algorithm.

Algorithm 1: Eve optimizer. Differences from Adam are highlighted: Eve becomes equivalent to Adam when using $\left(\frac{1}{B}\sum_{b=1}^{B} g_b\right)^2$ instead of $\frac{1}{B}\sum_{b=1}^{B} g_b^2$ in line 10, and omitting line 13. As discussed in section 3, lines 7 to 14 can be computed layerwise to lower runtime and peak memory. This can be implemented by leveraging compiler optimization and is not shown here for simplicity.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f$: Objective function with parameters $\theta$
Require: $batch$: Function returning a batch of training data for a given time $t$
Require: $\theta$: Initial parameter vector
1  $m \leftarrow 0$ (Initialize first moment vector)
2  $v \leftarrow 0$ (Initialize second moment vector)
3  $t \leftarrow 0$ (Initialize timestep)
4  while $\theta$ not converged do
5      $t \leftarrow t + 1$
6      $x_{1..B} \leftarrow batch(t)$ (Get next data batch)
7      for $b \leftarrow 1..B$ do
8          $g_b \leftarrow \nabla_\theta f_\theta(x_b)$ (Get examplewise gradient of objective)
9      $m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot \frac{1}{B}\sum_{b=1}^{B} g_b$ (Update biased 1st moment estimate)
10     $v \leftarrow \beta_1 \cdot v + (1 - \beta_1) \cdot \frac{1}{B}\sum_{b=1}^{B} g_b^2$ (Update biased examplewise 2nd moment estimate)
11     $\widehat{m} \leftarrow m / \left(1 - \beta_1^t\right)$ (Compute 1st moment estimate)
12     $\widehat{v} \leftarrow v / \left(1 - \beta_2^t\right)$ (Compute examplewise 2nd moment estimate, called $\widehat{v}_e$ in the text)
13     $\widehat{v} \leftarrow \frac{1}{B}\widehat{v} + \left(1 - \frac{1}{B}\right)\widehat{m}^2$ (Compute batch 2nd moment estimate)
14     $\theta \leftarrow \theta - \alpha\widehat{m} / \left(\sqrt{\widehat{v}} + \epsilon\right)$ (Update parameters)
15 return $\theta$
```

3. Evaluating Examplewise Gradients

The function $\overline{g}$ can usually be computed using automatic differentiation software. However, the sum on the right-hand side of eq. (3) is typically computed internally without exposing the summands to the user. One convenient way to access the 'examplewise gradient' terms $\nabla_\theta f(x_{d_b}, \theta)$ is to leverage a vectorizing map function. Assuming the cost function f, evaluated on a single element x and the parameters θ, has been defined, we can transform it by taking the gradient w.r.t. θ and applying a vectorized mapping over x. An example implementation using JAX is shown in listing 1. It is transferable to other AD libraries with support for compilation and automatic vectorization, such as PyTorch [13] and TensorFlow [1].

Listing 1: A training step implementation which uses JAX to evaluate the gradient of a function `f` on each element of a batch of data `xs`. The batch of gradient values in `grads` is used to update the parameters `theta`. The `jax.jit` decorator means that the `training_step` function will be just-in-time compiled the first time that it is called.

```
def f(x, theta):
    # <cost function for a single datapoint x>

@jax.jit
def training_step(xs, theta, opt_state):
    # Differentiate f w.r.t. the argument in position 1,
    # and create a closure over theta:
    grad_elem_f = lambda x: jax.grad(f, 1)(x, theta)

    # Evaluate the batch of gradients on the batch of data xs:
    grads = jax.vmap(grad_elem_f)(xs)

    # Use an optimizer update rule to update theta
    return optimizer.apply_gradients(opt_state, theta, grads)
```

Memory and runtime. Balles et al. [3] correctly note that storing B examplewise gradients in memory may be infeasible if the number of parameters is high. However, in our method, examplewise gradients for different layers never have to be in memory *at the same time*. Instead, they can be computed in sequence as part of backpropagation, and discarded once the desired statistics are extracted for each layer. For a sequential network with N parameters per layer, this results in additional peak memory usage of $O(BN)$, independent of the layer count. In our implementation, optimizations of the execution order of this type are automatically performed by the `jit` compiler. Our approach also exploits that all additional examplewise operations are not on the critical path of backpropagation, except for the first layer. The additional required runtime on parallel hardware with enough memory and processors is therefore constant w.r.t. the depth of the network, becoming insignificant for deep enough models. We found that in practice our implementation does have non-negligible runtime overhead for some tasks, details are in appendix B.

4. Experiments

In this section, we demonstrate the practicality of our proposed method to obtain examplewise gradients. We also give preliminary results comparing the performance of Eve to Adam.

Eve performs similarly to Adam on various tasks without retuning hyperparameters. We evaluate Eve and Adam on five tasks: CIFAR-10 image classification [11] with CNNs, addition with an LSTM [8] sequence-to-sequence model, transformer-based WMT machine translation [18], Atari Pong reinforcement learning based on PPO [17], and image modelling with PixelCNN++ [16]. We provide details in appendix B and full source code[1]. Preliminary results based on hyperparameters tuned for Adam are shown in table 1. We leave retuning Eve's hyperparameters for future work.

	CIFAR-10 acc.	Seq2Seq acc.	WMT acc.	Pong reward	PixelCNN log prob.
Eve (ours)	$69.0 \pm 1.0\%$	$98.4 \pm 0.0\%$	$70.8 \pm 0.04\%$	20.8 ± 0.2	-2.948
Adam	$68.0 \pm 0.5\%$	$98.4 \pm 0.0\%$	$71.4 \pm 0.11\%$	20.5 ± 0.4	-2.950

Table 1: Evaluation performance on various tasks. Higher is better. Averages and standard deviation over three runs are shown, except for PixelCNN with a single run.

Batch size	8	32	128	512
Eve	64.8%	71.3%	69.0%	69.4%
Adam	65.9%	72.0%	68.0%	66.2%

Table 2: Evaluation accuracy for various batch sizes on CIFAR-10 based on single runs.

Performance gain increase with larger batch size. Table 2 shows results for different batch sizes on CIFAR-10. While Adam performs better for smaller batches, Eve has an advantage over Adam for larger batch sizes.

Estimating examplewise variance only is insufficient. We evaluate Eve on CIFAR-10 without the batch variance correction on line 13 in algorithm 1. Evaluation accuracy drops drastically to 55.6%. Comparing the step size length for this ablation, Eve and Adam (fig. 1) shows that step sizes are much smaller, indicating the importance of the correction term.

Eve's step size is similar to Adam's but is more stable. Eve's standard deviation of the step size is 25% lower on CIFAR-10. Noise from stochastic gradients can help to escape bad local minima [10, 20]. Since Adam's less stable variance estimates translate into noisy step sizes, this might explain the performance lead of Adam over Eve on WMT. Further work is needed to test this hypothesis and understand when such noise is beneficial or harmful to learning.

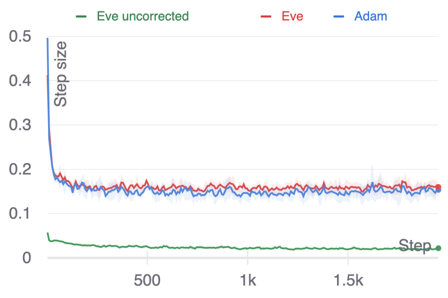

Figure 1: Parameter update step length over time on CIFAR-10.

Runtime and memory usage are practical even for large models. We report relative runtimes in section B. There is virtually no slowdown of Eve compared to Adam on the Seq2Seq tasks, which is expected due to the long critical path. Our experiments confirm that our approach is feasible in practice even for large models and batch sizes, such as WMT with 200M parameters and $B = 256$.

1. Code is available at `https://github.com/juliuskunze/eve`.

References

[1] Ashish Agarwal and Igor Ganichev. Auto-vectorizing TensorFlow graphs: Jacobians, auto-batching and beyond. *arXiv preprint arXiv:1903.04243*, 2019.

[2] L. Balles and P. Hennig. Dissecting Adam: The sign, magnitude and variance of stochastic gradients. In *ICML*, 2018.

[3] L. Balles, J. Romero, and P. Hennig. Coupling adaptive batch sizes with learning rates. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[4] Felix Dangel, Frederik Kunstner, and Philipp Hennig. BackPACK: Packing more into backprop. In *ICLR*, 2020.

[5] Ian Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

[6] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE conference on computer vision and pattern recognition*, 2016.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8), 1997.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[10] Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does sgd escape local minima? In *ICML*, 2018.

[11] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[12] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *ICLR*, April 2020.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 2019.

[14] Calyampudi Radhakrishna Rao. *Linear statistical inference and its applications*, volume 2. Wiley New York, 1973.

[15] Richard J Rossi. *Mathematical statistics: An introduction to likelihood based inference*. John Wiley & Sons, 2018.

[16] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. PixelCNN++: A PixelCNN implementation with discretized logistic mixture likelihood and other modifications. In *ICLR*, 2017.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[19] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[20] Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. The anisotropic noise in stochastic gradient descent: Its behavior of escaping from sharp minima and regularization effects. *arXiv preprint arXiv:1803.00195*, 2018.

Appendix A. Standard Error of Standard Deviation Estimator in Adam vs. Eve

The standard deviation ('standard error') of the population variance estimator $s^2 = \frac{1}{N}\sum_{i=1}^{N} g_b^2 - \left(\frac{1}{N}\sum_{i=1}^{N} g_b\right)^2$ for N i.i.d. normal samples $g_1, \ldots, g_N \sim \mathcal{N}\left(\mu, \sigma^2\right)$ is [15, Example 4.4, 4.12]:

$$\mathrm{SE}\left(s^2\right) = \frac{N-1}{N}\sqrt{\frac{2}{N-1}}\sigma^2 \approx \sqrt{\frac{2}{N}}\sigma^2. \tag{8}$$

The analog population variance estimator $\overline{s}^2 = \frac{1}{\overline{N}}\sum_{i=1}^{\overline{N}} \overline{g}_b^2 - \left(\frac{1}{\overline{N}}\sum_{i=1}^{\overline{N}} \overline{g}_b\right)^2$ applied to the $\overline{N} = \frac{N}{B}$ means $\overline{g}_1, \ldots, \overline{g}_{\overline{N}}$ of batches of size $B|N$, with $\overline{g}_i = \frac{1}{B}\sum_{b=1}^{B} g_{(i-1)B+b} \sim \mathcal{N}\left(\mu, \frac{1}{B}\sigma^2\right)$, has the standard error

$$\mathrm{SE}\left(\overline{s}^2\right) \approx \sqrt{\frac{2B}{N}}\frac{1}{B}\sigma^2 \overset{(8)}{\approx} \frac{1}{\sqrt{B}}\mathrm{SE}\left(s^2\right). \tag{9}$$

Using the delta method [14, pp. 386-391], we obtain an approximation of the standard error of the batch mean standard deviation estimators $\overline{s}$ and $\frac{1}{\sqrt{B}}s$ [15, Example 4.18]:

$$\mathrm{SE}\left(\frac{1}{\sqrt{B}}s\right) = \frac{1}{\sqrt{B}}\mathrm{SE}\left(s\right) \approx \frac{1}{2\sigma\sqrt{B}}\mathrm{SE}\left(s^2\right) \tag{10}$$

$$\mathrm{SE}\left(\overline{s}\right) \approx \frac{\sqrt{B}}{2\sigma}\mathrm{SE}\left(\overline{s}^2\right) \overset{(9)}{\approx} \frac{1}{2\sigma}\mathrm{SE}\left(s^2\right) \overset{(10)}{\approx} \sqrt{B}\mathrm{SE}\left(\frac{1}{\sqrt{B}}s\right). \tag{11}$$

Transferring this analysis to Adam and Eve, $\overline{\eta}$ and η have a relationship analogous to $\overline{s}$ and $\frac{1}{\sqrt{B}}s$, assuming a normal and slowly-changing gradient distribution. Then,

$$\mathrm{SE}\left(\eta\right) \overset{(11)}{\approx} \frac{1}{\sqrt{B}}\mathrm{SE}\left(\overline{\eta}\right), \tag{12}$$

so we expect that Eve's η standard deviation estimator is more stable than Adam's $\overline{\eta}$ by a factor of $\sqrt{B}$.

Appendix B. Task details

	CIFAR-10	Seq2Seq	WMT	Pong	PixelCNN
Batch size	128	128	256	256	320
Update steps	7k	2k	100k	400M	300k
Parameters	1.1M	2.2M	209M	4M	49M
Matmuls with parameters along critical path	4	512[A]	37[B]	6	36
Runtime of Eve relative to Adam[C]	120%	100%	129%	118%	454%[D]

Table 3: Task details. Notes: [A]Due to LSTM encoder and decoder rolled out for 256 steps each. [B]Only matrix multiplications with parameters are counted, not self-attention matrix multiplications. [C]Implemented in JAX, run on a TPU v3-8. [D]We suspect that this massive slowdown is caused by a bug.