From Adam to W

Jonathan Besomi
jonathan.besomi@epfl.ch

Stefano Huber
stefano.huber@epfl.ch

Yann Mentha
yann.mentha@epfl.ch

***Abstract*— As part of the *CS-439: Optimization for Machine Learning* course, we propose in the present document a comparative and practical study of three commonly used optimizers, namely SGD, Adam and AdamW to come up with an empirical estimation of their respective performances on common datasets and eventually reveal some significant difference across them. The followed approach aims notably at determining whether AdamW indeed offers superconvergence to a regular user with limited resource, in reference to the article "AdamW and Super-convergence is now the fastest way to train neural nets" [1].**

I. INTRODUCTION

The choice of the right optimizer plays an important role when training neural-networks. In our work, we compare three different stochastic optimizers, SGD with momentum and weight decay, Adam [2] and AdamW [3] on three different tasks. The main goal consists in comparing the robustness of each optimizer and its speed of convergence.

The optimizers are compared together on three different neural network tasks and datasets pairs: image classification on the MNIST dataset, speech classification on the Berlin Database of Emotional Speech dataset [4], and text classification on the AG_News dataset [5].

The present paper respects the following structure: section II give an overview of the optimizers, section III briefly discuss the different tasks, section IV sums up the main point of the code implementation and finally, sections V and VI describe the methodology of our experiments as well as our main findings.

II. OPTIMIZERS

A. Stochastic Gradient Descent (SGD)

SGD is an iterative method for optimizing an objective function. The optimal values is found by computing the gradient and by following it's opposite direction. We used SGD with momentum, that leads in general to faster convergence by means of the computation of a moving average of the gradients. Intuitively, this reduces the amount of noisy updates since if one update is noisy, its noise is averaged out by the previous history of updates (the momentum). A step of the gradient descent with momentum is defined as:

$$v_{t+1} = \gamma v_t + g_{t+1} \; ; \; w_{t+1} = w_t - \lambda v_{t+1}$$

Where g_t, w_t, v_t, λ, γ are respectively the gradient of the loss function at time t, the weights, the velocity, the learning rate and the momentum. [1]

B. Adam

Adam stands for adaptive moment estimation. Adam is the combination of two extensions of stochastic gradient descent with momentum, AdaGrad [7] and RMSProp[2] [8].

1) Adagrad: With Adagrad, the learning rate is adapted to the parameters. Low learning rates is applied for noisy parameters (which have a high second moment, which is non centered approximation for the variance), and large learning rates are applied for the parameters updated less frequently. The update step rules read as:

$$w_{t+1} = w_t - \frac{\lambda}{\sqrt{diag(G_t + I\epsilon)}} \odot g_t$$

where ϵ is a small quantity to avoid zero-divisions, I is the identity matrix and $G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^T$ is a matrix where every diagonal element (i, i) is the sum of the squares of the gradient g_i.

Adagrad's main drawback is that it acculumates the squared gradient in the denominator. As each term is positive, the accumulated sum will eventually grow and this will cause the learning rate to become infinitesimally small.

2) RMSprop: RMSprop tries to solve the dissolving learning rate issue with a moving average (by means of a decaying factor) on the squared gradient sum. That way, the sum does not grow out of control.

$$diag(G_t + I\epsilon) = \beta diag(G_{t-1} + I\epsilon) + (1 - \beta) g_t^2$$

[1]The displayed momentum formula corresponds to the one used in the SGD PyTorch implementation. This subtly differs from the definition of the classical momentum [9]

[2]RMSProp has been proposed by Geoff Hinton on his Coursera Course. In the same period, Adadelta [6], an alternative RMSProp was also published.

3) Adam: Adam takes advantage of both of the aforementioned strategies and stores two exponentially decaying averages of past squared gradients (similarly to RMSprop) and of the gradients (similarly to standard momentum).

$$m_t = B_1 m_{t-1} + (1 - B_1) g_t$$
$$v_t = B_2 v_{t-1} + (1 - B_2) g_t^2$$

m_t and v_t are initialized as zero-vectors and therefore biased towards zero. The authors counteracts these biases by computing bias-corrected first and second moment estimates:

$$like\hat{v}_t = \frac{v_t}{1 - B_2^t} \; ; \; \hat{m}_t = \frac{m_t}{1 - B_1^t}$$

where B_1 and B_2 are numbers close to 1 (like 0.9, 0.999) and therefore the division by $1 - B_1^t$ and $1 - B_2^t$ tends to be a division by 1 when the training episode t gets large.

A gradient descent step in Adam is hence defined as follow:

$$w_{t+1} = w_t - \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

C. AdamW

With AdamW, a L2 regularization factor is introduced on top of Adam. L2 regularization is a well-known method to reduce over-fitting and consists of adding sum of the squares of all the weights as penalty term to the loss function (multiplied by a constant known as weight decay).

$$loss = loss + weight\_decay \frac{1}{2} \sum w_i^2$$

The constant factor is called weight_decay as in the gradient step rules the weights are *decayed* by that factor.

$$w_{t+1} = w_t - \lambda g_t - \lambda * weight\_decay * w_t$$

L2 and weight_decay are the same thing only in case of pure SGD. When we add momentum, for instance, the two functions are not anymore the same[3]. AdamW treats therefore differently L2 regularization and weight_decay, and implements the second equation. We also note that in our PyTorch implementation, both L2 regularization and *weight_decays* parameters are called in the same manner even if they are implemented differently.

[3]See the first formula under SDG

III. Three tasks

A. Image classification

For the image task we decided to use the MNIST dataset since it is both one of the most famous benchmark for image processing, and still small enough to be handled by limited hardware (single GPU) in reasonable time. The task consists in classifying hand written digits from 0 to 9, on images of 28x28 grey-scale pixels. The model chosen is a single CNN, which achieves a validation accuracy of $\approx 99\%$.

B. Text classification

The task consist in classifying news in four different topics. The total number of training samples is 120000, 30000 training sample for classes and the testing sample is $7,600$. The dataset is loaded directly from torchtext, a PyTorch sub-module to work with text data. The initial comes already pre-processed, tokenized and vectorized[4]

The final model consists simply of an embedding layer of 64 dimension followed by a linear layer. The model achieves an accuracy of approximately 92% on validation set. We tested the dataset on a more complex architecture with one and two layers of recurrent neural network (both GRU and LSTM). A single epoch just took much longer ($>$ 1 minutes on GPU) and the accuracy was about the same (if not worse) and therefore we opted for the simple model.

C. Speech recognition

The task consists in classifying various emotion voice samples in german: the same sentences are read by comedian with 6 types of tone (angry, sad, happy, disgusted, affraid and bored) and recorded in .wav files. As most of every state-of-the art speech recognition task, the sound features are converted to their mel-frequency cepstrum (mfcc), since this type of transformation conserves and exhibits accurately most of the human voice-specific frequence ranges. These mfcc features, which can be visualized as an image since it consists of 2d tensor, gets ideally classified by CNNs architecture and LSTMs. The chosen model consists of a deep-CNN architecture as described in [16]. Its training accuracy reaches $\approx 96\%$ for $\approx 65\%$ of validation accuracy.

[4]Retrospectiy, it would have been better to have finer control on the initial dataset, rather than selecting out-of-the-box solutions, as this allows to control better the experiment.

IV. Implementation

A. Pure PyTorch

Our code implementation is mainly based on PyTorch. The structure of the project has been conceived so that as much code as possible is shared between the three tasks. For details regarding the code and its structure, the reader can refer to the `README.md` in the root of the repository as well as the docstring of each function.

B. Hyperparameter optimisation

1) Weight initialization: As our primarily goal consist in comparing the performance of optimizers, all layers of the neural network are initialized with the default settings, assuming that in most cases this values are kept unchanged by the practitioners.

2) Batch normalization: Batch normalization is generally used to increase the stability of a neural network. In our solution, we didn't introduce any batch normalization to facilitate the interpretation of the results across different optimizers and limit the differences across them.

3) Cross validation: In order to select the best parameters, we experimented both with K-fold cross validation and repeated random train-validation sets. The second option was preferred, as K-fold turns out to be too computationally costly. Each time, the combinations of parameters displaying the highest average accuracy (after early stopping) are picked.

4) Early stopping: This feature was added to limit overfitting: during training, the validation loss is measured at each epoch. If it doesn't decrease for a certain number of epochs (called patience) the training stops. Otherwise it continues up to the specified number of epochs for that task. This mechanism is systematically performed during cross validation, for each parameter combination and train-validation split.

C. Training with GPU

All tests have been conducted on machine with GPU capabilities for faster training. We used both our own machine with GPU as well as Google Colab.

D. Reproducibility

We started with the idea that we wanted all our results to be 100% reproducible just with a single click. This would mean that all experiments must be done sequentially, starting from a fixed seed. In practice, though, we had to speed up the process and therefore executed the experiments for each task in parallel. Nonetheless, calling twice `python main.py` on the same machine produce the same exact results.

E. Global metric

As all three tasks are multi-class classification and each of them display balanced classes, the selected metric is the accuracy for facilitate comparison.

V. EXPERIMENTS

A. Student Test

In order to test the significance of the difference in terms of loss function between the two optimizers Adam and AdamW, we decided to fit an exponential curve on their respective training and validation losses throughout the grid search using least squares in the lin-log or log-log space, depending on the shape of the data (cf Fig 4 (lin-log) and 1 (log-log)).

Although all of its statistical assumptions were not met, the student t-test was implemented and used to get an intuitive insight on these curves by computing a p-value for the slopes difference: this statistic is therefore to be taken with a grain of salt. The common assumptions made when doing a t-test include:

1) The scale of measurement (met)
2) Random sampling (not met)
3) Assumption of normality for the data distribution (not tested)

Note that in the present experiment, a particular focus was put on the optimizer convergence speed rather than actual model performance (although one of these poor model performances impairs convergence interpretability as we will see later). In that sense, an optimizer A that reduces quicker the loss than another optimizer B is considered as more performant even if a clear overfit phenomenon is observed (initial reduction of the validation loss before increasing again). Indeed, both of the optimizer jobs consist of going down in the train loss landscape, no matter if the optimum of this landscape generalizes well to other datasets.

VI. RESULTS

A. Images classification

The image dataset displayed a significant preference for the AdamW optimizer both on training and validation sets over Adam. Although the mean curve might at first sight looks discontinuous, this is in fact due to early stopping: as the mean averages the values of the existing curves and some curves stops at given epochs, this explains the sudden jumps of the signal. Note that AdamW seems to find a deeper optimum more regularly than Adam in the training loss landscape.(Fig 1).

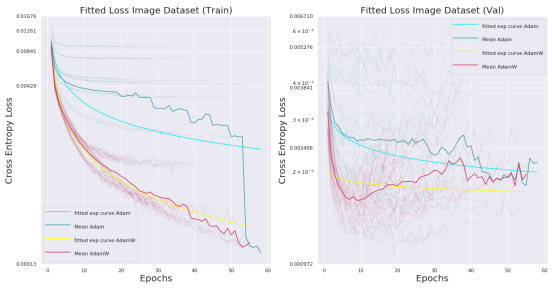

Fig. 1. Training loss and validation loss for the Image dataset: a pvalue of 0.04 was achieved for the t-test on the slope difference of the fitted linear regression in the loglog space.

B. Text classification

The text dataset revealed the most disparate results across optimizers as shown on Figure 2 with AdamW displaying the best results, leading to a rapid overfit. We note two distinct behaviors for AdamW on the validation plot: the first clearly diverges whereas the second shows the typical overfitting pattern.

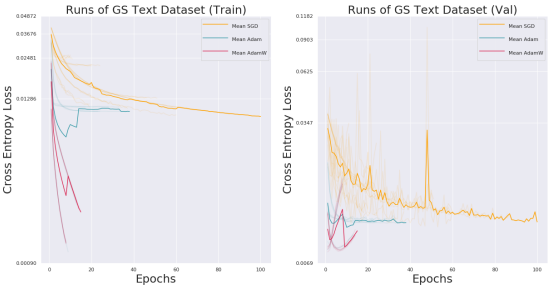

Fig. 2. Grid search: train vs validation on text dataset

A specific analysis of these validation losses reveals that the learning rate segregates between the two behaviors: when set to 0.01, it diverges in terms the validation whereas setting it to a smaller value allows one to reach a local optimum before diverging.

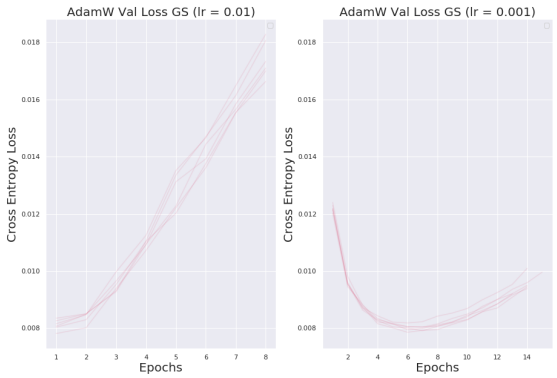

Fig. 3. Validation Loss for various lrs: as a high lr immediately overfit and increases the validation loss, smaller lr lead to overall better results on the present dataset.

C. Speech classification

No significant difference has been observed between Adam and AdamW on the speech dataset. However, they both reduce the training loss significantly faster than SGD, even with momentum. Notice the algebraic convergence on this dataset for all optimizers, as the loss tends to decrease linearly in the lin-log space. A p-value of 7.41E-10 has been computed on validation set between the Adam and SGD regressions (in the lin-log space). This goes along with a quick overfit and therefore learning speed in favor of the Adam optimizers. However, probably due to a poor model architecture or due to a too-small dataset, the validation loss does not decrease significantly for any of the optimizers, making the speech dataset less trustful for the optimizers evaluation. (cf Figure VI-D, where the validation and test loss exhibit unstable behavior)

D. Adam vs W

As an overall result, it seems that AdamW indeed allows to perform a slightly quicker convergence than the Adam counterpart and SGD, as shown on the 3 training sets of Figure VI-D (A1, B1, C1).

The same seems to hold true for the validation set in the present experiment, provided that the model architecture generalizes decently. (A2, B2)

Note however that only the text test set conserves this behavior (A3). This might be due to some ”hyper-overfitting” where the selected parameters during the

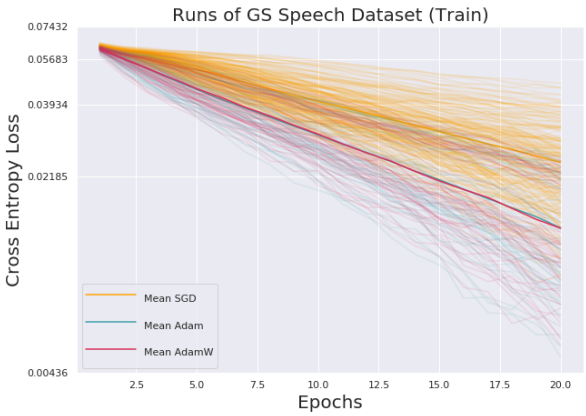

Fig. 4. Training loss for the Speech dataset: the loss decrease algebraically as it exhibits linear behaviour in the lin-log space. Although indistinguishable between each other, Adam and AdamW both perform significantly better than classical SGD even with momentum. P-value between SGD-regression slope and Adam-reg slope in the lin-log space: 7.41E-10

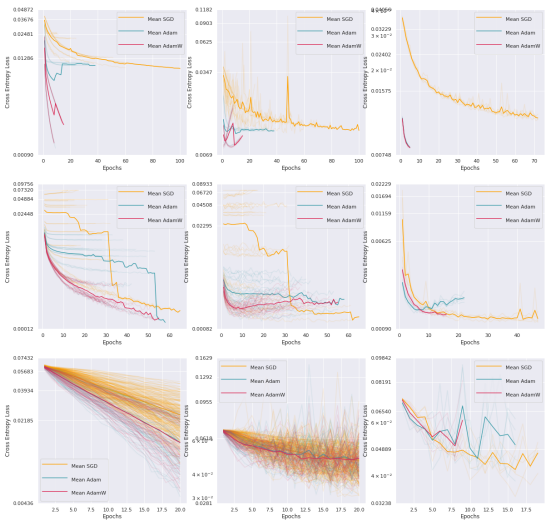

Fig. 5. An overall view of the experiments: in order the Text, Image and Speech datasets (rows: A-B-C) for their respective train, validation and test losses (columns:1-2-3). A1, B2, B3, A2, B2 and A3 show significantly favorable results for Adam and/or AdamW (t-test for the slope) whereas the rest does not show worse performance (non-significant)

grid search are the one which performs well on the training set exclusively. Running the experiment on other datasets could confirm this theory and would be relatively inexpensive as the framework necessary to test it is now implemented.

VII. CONCLUSION

For three different neural network tasks and their respective datasets, we implemented and compared different optimizers; the well-known SGD with momentum and weight decay, Adam, an optimization algorithm that adapts the learning rate according to the weights frequency update and AdamW, a variation of Adam where the L2 regularization is replaced by weight decay.

We developed a custom-yet-flexible implementation of the code to allow for efficient testing on multiple tasks and tested the optimizers extensively. Although we tried to tune as much as possible the parameters such as the model architecture (to permit fair comparison) and extensively searched for the best parameters of each optimizers, we faced an expected difficulty at demonstrating a significant dominance of one optimizer over the others.

Rather, we highlighted trends that seem to go for the majority in the sense of the article that motivated the present document, although the promised superconvergence results seemed a bit overemphasize as they don't generalize systematically: AdamW shows indeed slightly better performances, which at large scale could potentially relieve some computational constraints.

However a lambda user would not necessarily notice the difference with Adam on an every-day coding basis: the biggest gap between two optmizers remains the one separating Adam from SGD. This motivates the need for further investigations on popular datasets and development of more elaborated methods to compare optmizers in the future, and benchmark more precisely the performances of optimizers such as AdamW.

REFERENCES

[1] Sylvain Gugger and Jeremy Howard, AdamW and Super-convergence is now the fastest way to train neural nets, https://www.fast.ai/2018/07/02/adam-weight-decay/
[2] Diederik P. Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization
[3] Ilya Loshchilov, Frank Hutter. "Decoupled Weight Decay Regularization" arXiv:1711.05101
[4] Berlin Database of Emotional Speech, Berlin Database of Emotional Speech
[5] G. M. Del Corso, A. Gulli, and F. Romani. Ranking a stream of news. In Proceedings of 14th International World Wide Web Conference, pages 97–106, Chiba, Japan, 2005.
[6] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. http://arxiv.org/abs/1212.5701

[7] John Duchi, Elad Hazan, Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. 12(61):2121 2159, 2011

[8] Geoff Hinton in Lecture 6e of his Coursera Class, lecture slides

[9] Polyak, B.T. Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5):1–17, 1964.

[10] LeCun, Yann and Cortes, Corinna. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2010.

[11] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[12] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[13] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32 (pp. 8024–8035). Curran Associates, Inc..

[14] Sebastian Ruder (2016). An overview of gradient descent optimisation algorithms. arXiv preprint arXiv:1609.04747.

[15] Ilya Sutskever, James Martens, George Dahl and Geoffrey Hinton. On the importance of initialization and momentum in deep learning, http://www.cs.toronto.edu/ hinton/absps/momentum.pdf

[16] Venkataramanan, Kannan, and Haresh Rengaraj Rajamohan. "Emotion Recognition from Speech." arXiv preprint arXiv:1912.10458 (2019).