Performance of Adaptive Stochastic Gradient Descent (AdSGD) on simple model and standard datasets

Louis Jaugey : louis.jaugey@epfl.ch
Haojun Zhu : haojun.zhu@epfl.ch
Aurelio Noca : aurelio.noca@epfl.ch,
EPFL, Switzerland

***Abstract*—The field of Machine Learning has shown to be extremely effective in a vast panel of applications. Training these algorithms is an extremely difficult optimization problem and many minimization techniques have been devised and applied to neural networks. However most of those methods require some arbitrary parameters (hyperparameters) which must generally be guessed by trial and error. Reducing optimizations methods' sensitivity to these hyperparameters is therefore particularly interesting since it would allow to use a single method that could generalize to various network architectures.**

In this work, we investigate a new optimization method called adaptive stochastic gradient descent (AdSGD), which is guaranteed to converge with relatively weak assumptions by adapting the learning rate dynamically. We compare the optimization method on 4 datasets using a simple model with the results from the original paper [1] and verify their claims.

I. Introduction

Being one of the most useful and fundamental optimization methods, Gradient Descent (GD) has its limitations (choice of stepsize λ, robustness, convergence rate, etc). In a recent work [1], Y.Malitsky and K.Mishchenko proposed a new variant of GD, namely the Adaptive GD to overcome the aforementioned issues. In their experiments, the Adaptive SGD optimizer, which is an extension of their method, shows better performance in terms of convergence rate and test accuracy against some popular optimizers such as Adam in an image classification task.

In this project, we mainly investigate the performance of the Adaptive SGD algorithm on different datasets (MNIST, EMNIST, Fashion-MNIST, and CIFAR10) with a lighter neural network model, due to the limited computing power. Furthermore, we compare the results from Adaptive SGD to two different optimizers, Adam and momentum version of SGD (SGDm). Finally, we discuss the claims made by the authors of the paper about AdSGD.

II. Optimizer

The Adaptive GD algorithm proposed in [1] is described in algorithm A.1. This optimizer is extended to a stochastic gradient descent scenario, referred to as AdSGD. It uses an estimate of the Lipschitz constant of the function to compute the learning rate. The method is specified in algorithm 1.

Algorithm 1 Adaptive Stochastic Gradient Descent

Input: $x^0 \in \mathbb{R}^d, \lambda_0 > 0, \theta_0 = +\infty, \xi^0, \alpha > 0$
$x^1 = x^0 - \lambda_0 \nabla f_{\xi^0}(x^0)$
for k = 1,2,... **do**
Sample ξ^k and optionally ζ^k
Option I (biased): $L_k = \frac{\|\nabla f_{\xi^k}(x^k) - \nabla f_{\xi^k}(x^{k-1})\|}{\|x^k - x^{k-1}\|}$
Option II (unbiased): $L_k = \frac{\|\nabla f_{\zeta^k}(x^k) - \nabla f_{\zeta^k}(x^{k-1})\|}{\|x^k - x^{k-1}\|}$
$\lambda_k = \min\{\sqrt{1+\theta_{k-1}}\lambda_{k-1}, \frac{\alpha}{L_k}\}$
$x^{k+1} = x^k - \lambda_k \nabla f_{\xi^k}(x^k)$
$\theta_k = \frac{\lambda_k}{\lambda_{k-1}}$
end for

Note that ξ_k is a minibatch sample. The algorithm offers two options to estimate L_k in order to determine λ_k. The first option is to directly use $\nabla f_{\xi^k}(x^k)$ to compute L_k at the k-th iteration, which makes $\lambda_k \nabla f_{\xi^k}(x^k)$ biased; the alternative option is to estimate L_k from another minibatch sample ζ^k by the corresponding stochastic gradient $\nabla f_{\zeta^k}(x^k)$. From the previous experiments conducted in [1], the first option gives better performance.

III. Experiments

A. Datasets

Four standard datasets are used to test the Adaptative SGD algorithm:

- MNIST: A dataset of handwritten digits from 0 to 9 in grayscale. Images of size $1 \times 28 \times 28$. Training set: 60'000 samples. Test set: 10'000 samples.
- FMNIST: A dataset of 10 classes of pictures of clothing in grayscale. Images of size $1 \times 28 \times 28$. Training set: 60'000 samples. Test set: 10'000 samples.
- EMNIST: A dataset of handwritten letters from a to z in grayscale. There are 26 classes. Images of size $1 \times 28 \times 28$. Training set: 124'800 samples. Test set: 20'800 samples.
- CIFAR10: A dataset of 10 different classes in color. Images of size $3 \times 32 \times 32$. Training set: 50'000 samples. Test set: 10'000 samples.

B. Model

The experiments in [1] are conducted with ResNet-18 and DenseNet-121 on the CIFAR10 datasets. Due to the limitation of computing resources, we test the Adaptive SGD method on a custom modified version of the LeNet5 architecture [2]. The modified architecture is described in table I. The activation functions are replaced by ReLU and the cross-entropy loss is utilized. Furthermore, due to the properties of the different datasets, the model is slightly modified to allow for the input/output sizes to correspond to the sizes of the images/labels. Letter a corresponds to the MNIST dataset, b to the FMNIST dataset, c to the EMNIST dataset and d to the CIFAR10 dataset.

TABLE I
ARCHITECTURE AND CHARACTERISTICS OF THE MODIFIED LENET5

Layer	Layer name	Characteristics
1.a/b/c	Conv2d	1 input channel, 6 5x5 kernels with padding 2
1.d	Conv2d	3 input channels, 6 5x5 kernels with padding 2
2	ReLU	-
3	AvgPool2d	2x2 kernels, stride 2
4	Conv2d	6 input channels, 16 5x5 kernels
5	ReLU	-
6	AvgPool2d	Kernels of size 2x2, stride 2
7	Flatten	-
8.a/b/c	Linear	Input size: 400, output size: 120
8.d	Linear	Input size: 576, output size: 120
9	ReLU	-
10	Linear	Input size: 120, output size: 80
11	ReLU	-
12.a/b/d	Linear	Input size: 80, output size: 10
12.c	Linear	Input size: 80, output size: 26

C. Hyperparameters of AdSGD

In practice, the learning rate λ_k is updated in a slightly different way, compared to the formula in Algorithm 1. In order to reach better performance, two hyperparameters, the learning rate amplifier γ and learning rate damping $\delta = \frac{1}{\alpha}$, are introduced to tune the Adaptive SGD optimizer. The empirical update of the learning rate is as follows:

$$\lambda_k = \min\{\sqrt{1+\gamma\theta_{k-1}}\lambda_{k-1}, \frac{1}{\delta L_k}\}$$

D. Training experiments

The model is trained on 100 epochs using batch sizes of 512. The training experiments are as follows:

- Adam : using Adam optimizer with standard values, i.e. initial learning rate $\lambda = 0.001$.
- SGDm : using SGD with momentum $\beta = 0.9$ and learning rate $\lambda = 0.01$.
- AdSGD: using initial learning rate $\lambda = 0.2$ and parameters $(\gamma, \delta) = (0.1, 1)$.
- AdSGD: using initial learning rate $\lambda = 0.2$ and parameters $(\gamma, \delta) = (0.02, 1)$.
- AdSGD: using initial learning rate $\lambda = 0.2$ and parameters $(\gamma, \delta) = (0.02, 2)$.

Each experiment is repeated 5 times, in order to obtain a confidence interval of the results, one standard deviation in this case. All the graphs presented in the section IV show the average value in full color and one standard deviation around it in light color. All the experiments are run using Pytorch, on the Izar cluster[1] in order to take advantage of GPUs (NVIDIA V100 32 GB). The code for the AdSGD algorithm [3] was modified and adapted to run these experiments.

IV. RESULTS AND DISCUSSION

First the three AdSGD experiments are performed. The test accuracy on CIFAR10 is shown in fig. 1, and the other results are available in the appendix (fig. A.1, fig. A.2, fig. A.3, fig. A.4).

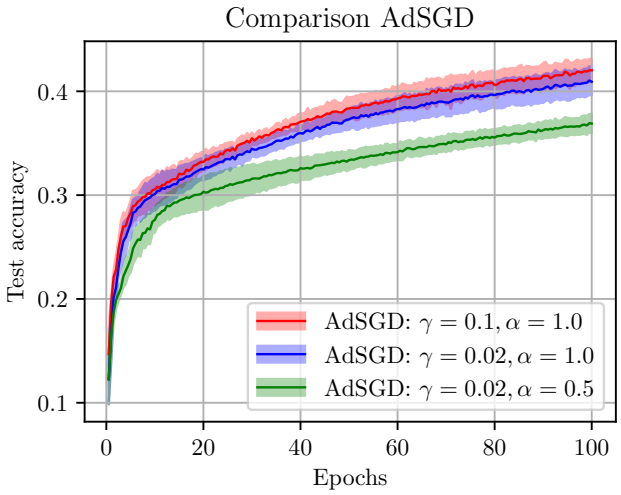


Fig. 1. Test accuracy vs number of epochs for the three AdSGD experiments on the CIFAR10 dataset.

The optimizer with $(\gamma, \delta) = (0.1, 1.0)$ is the one that performs the best on all the datasets. The performance seems to improve for larger learning rate amplifiers γ and smaller damping δ. Overall, this leads to choosing larger learning rates. In general, larger steps seem to allow for faster convergence for the chosen set of experiments. Interestingly, all three experiments seem to keep improving at the 100^{th} epoch, which indicates that training for longer could lead to better results. However, we trained the best AdSGD for 200 epochs and the results only got marginally better. Due to the limited computation power and results improvement, the number of epochs was kept at 100.

The results of the best AdSGD optimizer are compared to Adam and SGDm in the appendix for the first three datasets (fig. A.5, fig. A.6, fig. A.7). The CIFAR10 dataset is investigated in more details with fig. 2 which show the test accuracy, the training loss and the evolution of learning rates.

For the first few epochs, one can observe that the AdSGD method performs better than the SGDm optimizer, but the latter quickly overtakes it. The rate of increase for the adaptive method's test accuracy slows down considerably after the early epochs. From the graph of the training loss, it appears the optimizer is reaching a local minimum as the loss plateaus. In

[1]Running the experiments sequentially were vastly under-utilizing the GPU. Therefore, the 5 experiments were run concurrently using 5 MPI tasks.

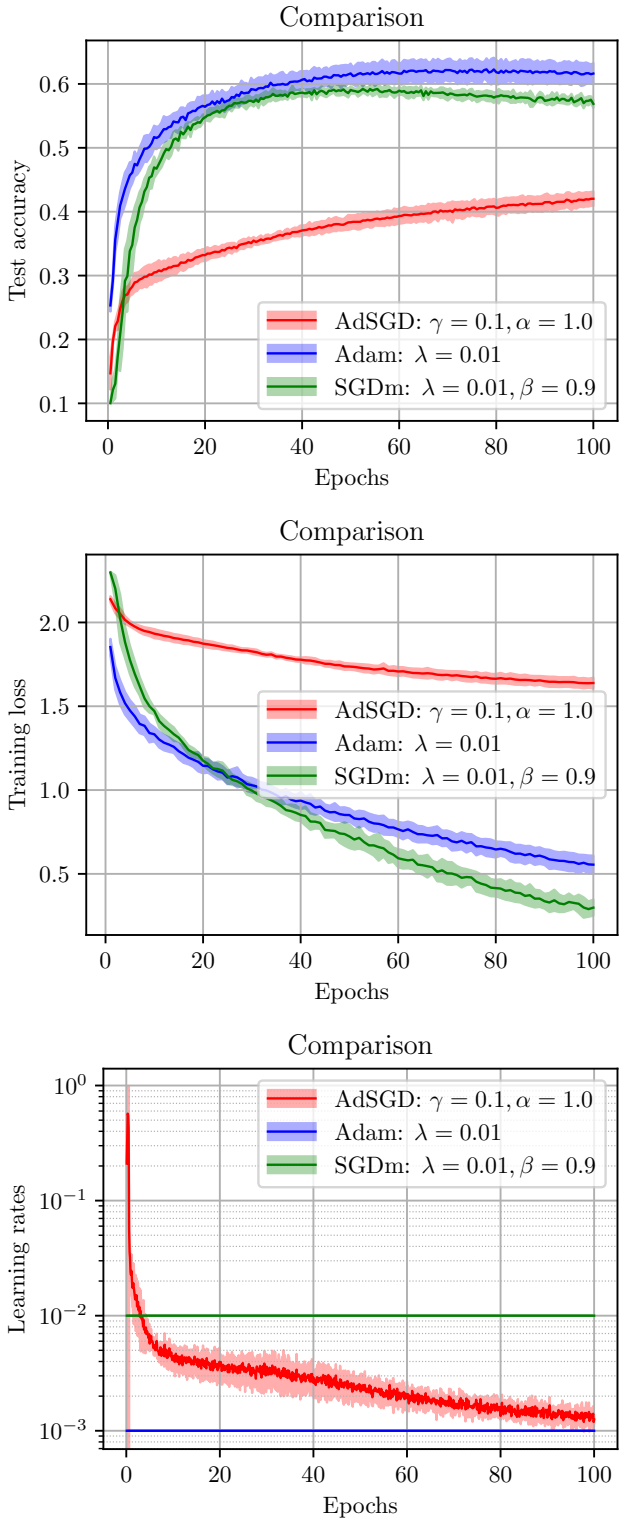


Fig. 2. Training loss (top), learning rates (middle) and test accuracy (bottom) vs number of epochs using Adam, SGDm and AdSGD on the CIFAR10 dataset.

contrast, the training losses of the SGDm and Adam optimizers plummets in comparison. After the 30^{th} epoch, SGDm has a lower training loss than Adam. It seems the former starts to overfit, as can be seen from a slight rate of decrease in the test accuracy. Interestingly, though the AdSGD performs worse than both other optimizers, apart from the initial large value, the learning rate is comprised between the learning rates of SGDm and Adam. It may be interesting, as a future endeavour, to compare the gradients obtained with the AdSGD method with the moments of the Adam method.

The obtained results are not in line with the ones from the original paper, where they showed that AdSGD was converging faster and to better optima. There could be multiple potential explanations to this difference. One of them is the batch size. Indeed, they used a batch size of 128 whereas we used 512[2]. We tried to use a batch size of 128 and observed a notable improvement. However, the other optimizers also improved slightly and were still better than AdSGD. It seems like AdSGD is still very sensitive to hyperparameters. It is possible that choosing other combinations of the learning rate amplifier and damping could yield better results for the method. However this would mean extensive hyperparameter search would need to be performed, which would refute the idea of a "free lunch" the authors claim to offer with this new optimizer.

Another potential explanation is that AdSGD works better for large models. This claim is however harder to verify, again because of the limited computation resources.

V. Conclusion

In this work we implemented a simple neural network and trained it on several standard datasets – MNIST, FMNIST, EMNIST and CIFAR10 – using the novel Adaptive Stochastic Gradient Method. Experiments were done to compare different choices of hyperparameters for this method. The testing accuracy of the best method was compared to other standard optimizers. The performance of AdSGD was worse on all the datasets, potentially due to the method converging to a local minimum. Therefore, this work was not able to reproduce the positive results from [1] on this set of experiments. Although our results may be improved by further fine-tuning, this report shows that the method still requires a deliberate choice of the hyperparameters to ensure satisfying performance.

References

[1] Y. Malitsky and K. Mishchenko, "Adaptive gradient descent without descent," 2019. [Online]. Available: https://arxiv.org/abs/1910.09529

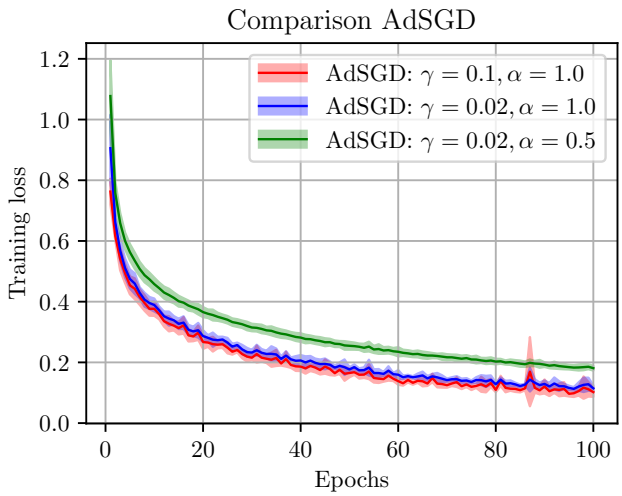
[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

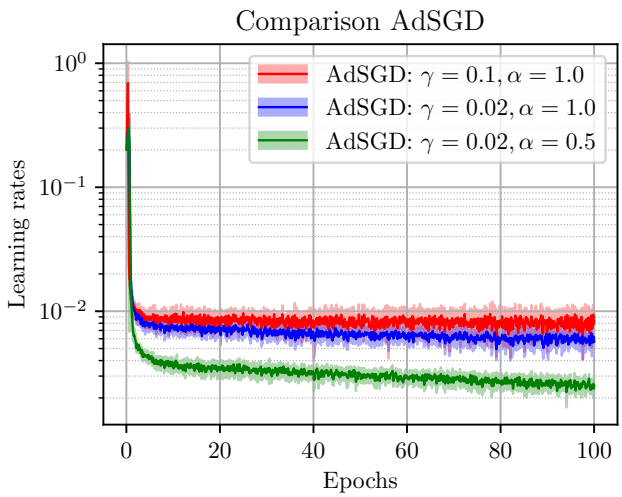
[3] Y. Malitsky, "adaptive_gd," https://github.com/ymalitsky/adaptive_GD, 2021.

[2]A batch size of 512 was chosen simply because it was the fastest to train on our hardware.

APPENDIX

Additional figures





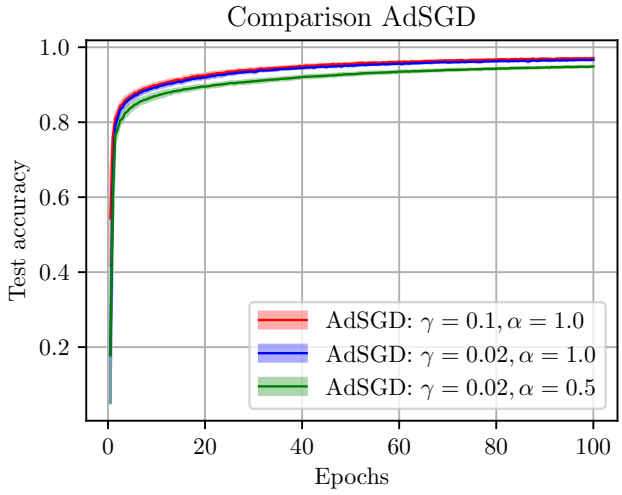
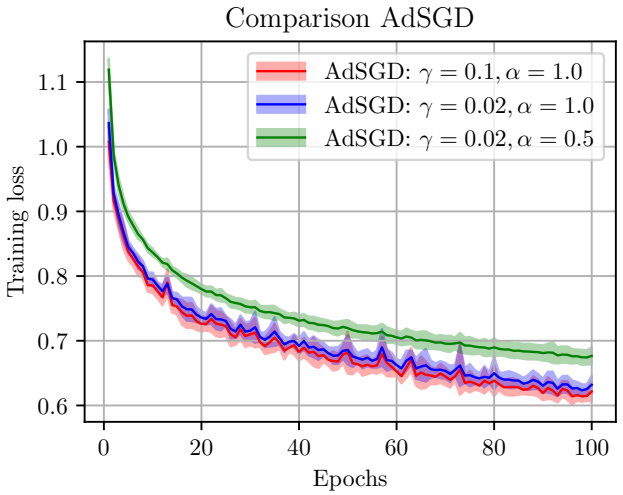
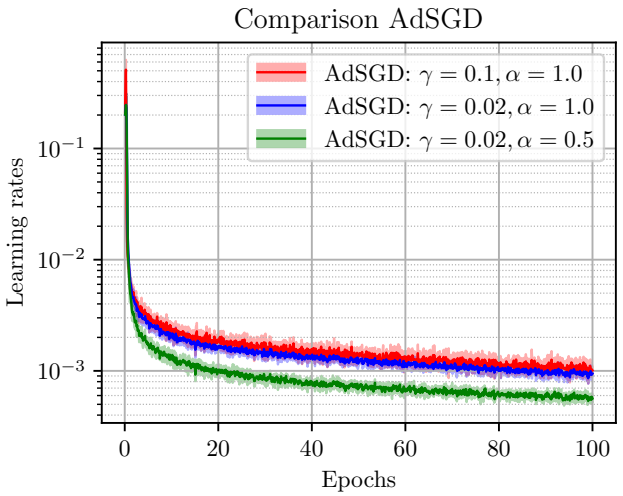


Fig. A.1. Training loss (top), learning rates (middle) and test accuracy (bottom) vs number of epochs for the three AdSGD experiments on the MNIST dataset.





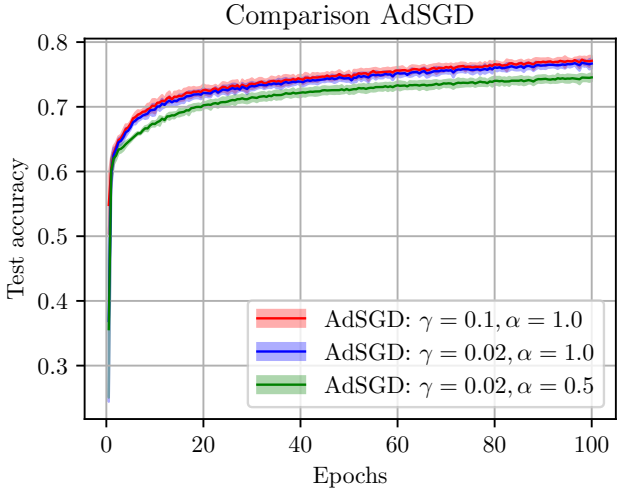
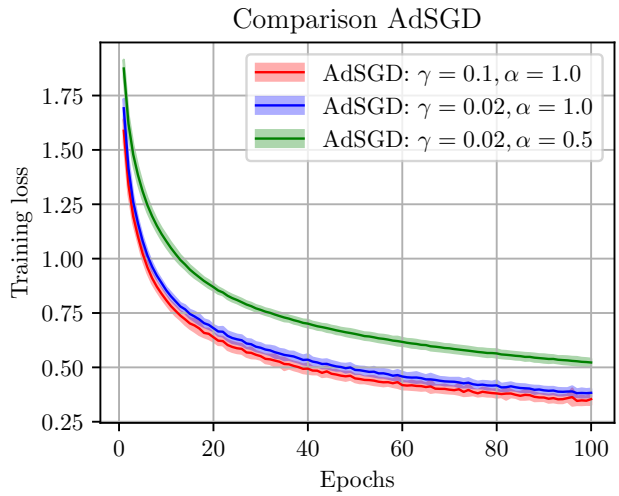
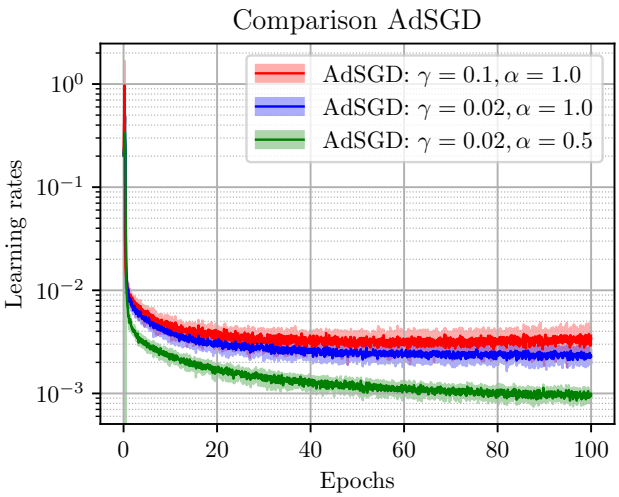


Fig. A.2. Training loss (top), learning rates (middle) and test accuracy (bottom) vs number of epochs for the three AdSGD experiments on the FMNIST dataset.





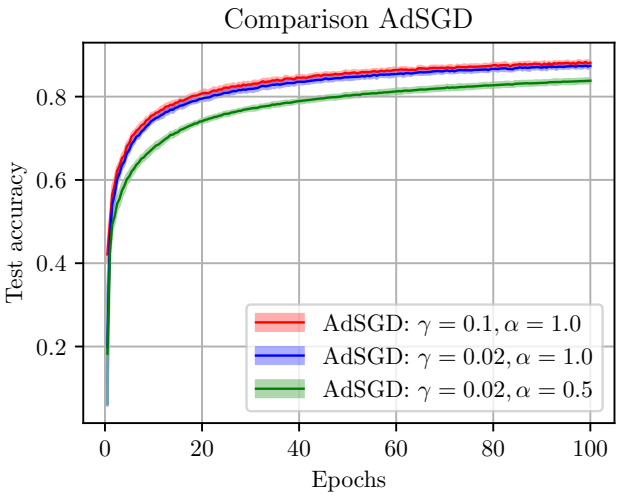
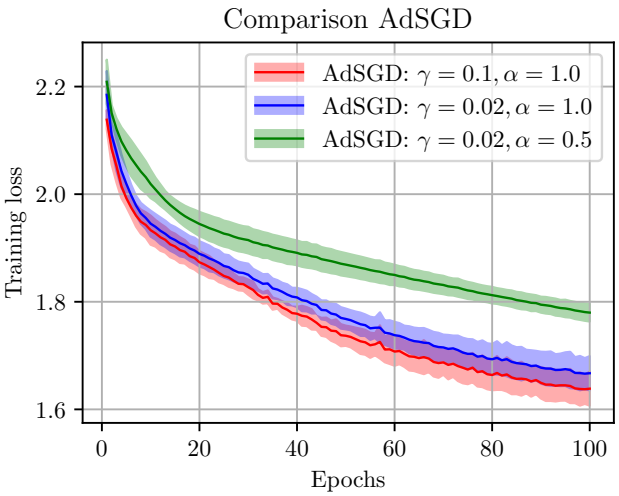
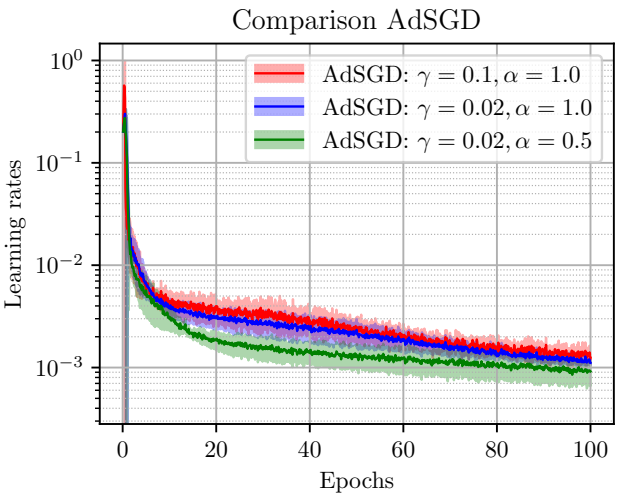


Fig. A.3. Training loss (top), learning rates (middle) and test accuracy (bottom) vs number of epochs for the three AdSGD experiments on the EMNIST dataset.





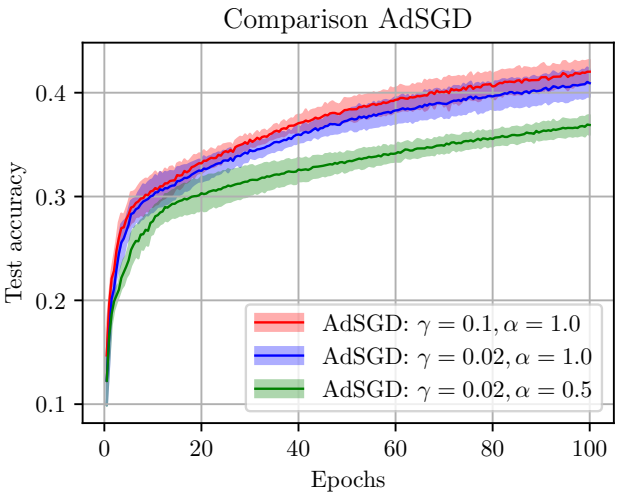
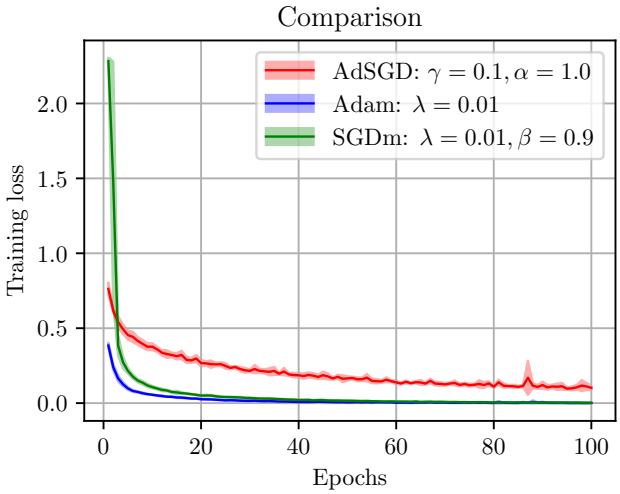
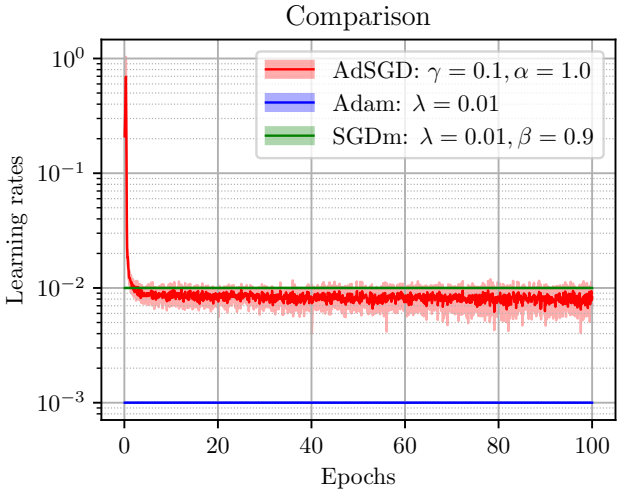


Fig. A.4. Training loss (top), learning rates (middle) and test accuracy (bottom) vs number of epochs for the three AdSGD experiments on the CIFAR10 dataset.





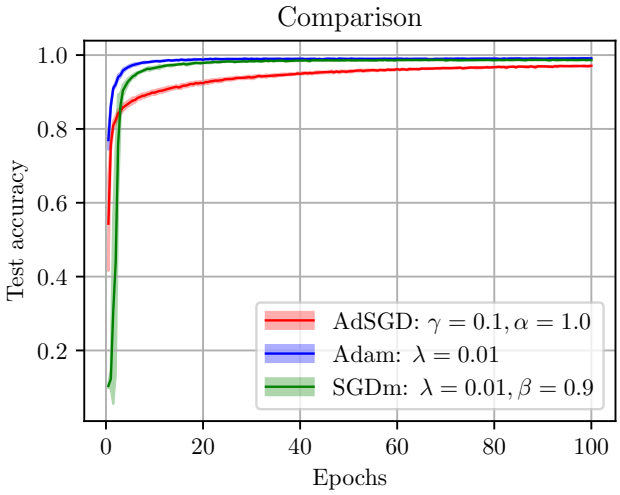
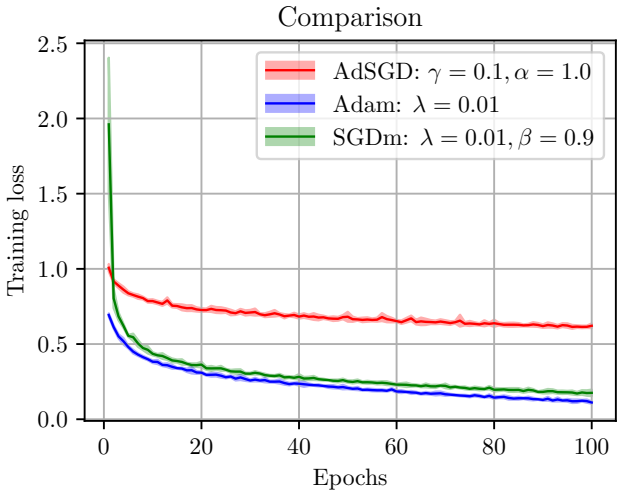
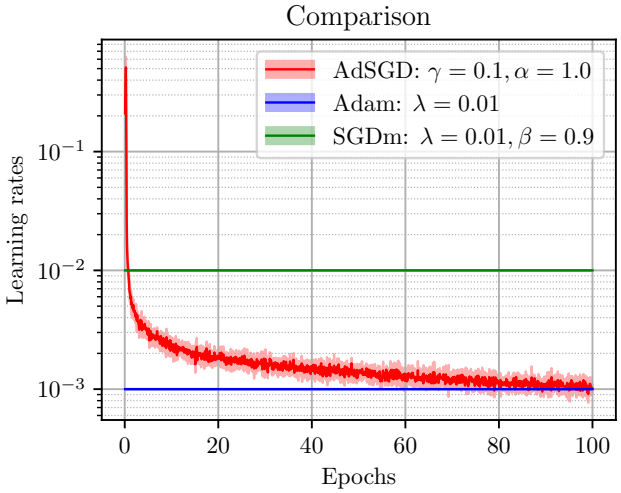


Fig. A.5. Training loss (top), learning rates (middle) and test accuracy (bottom) vs number of epochs for the experiments with the AdSGD, Adam and SGDm optimizers on the MNIST dataset.





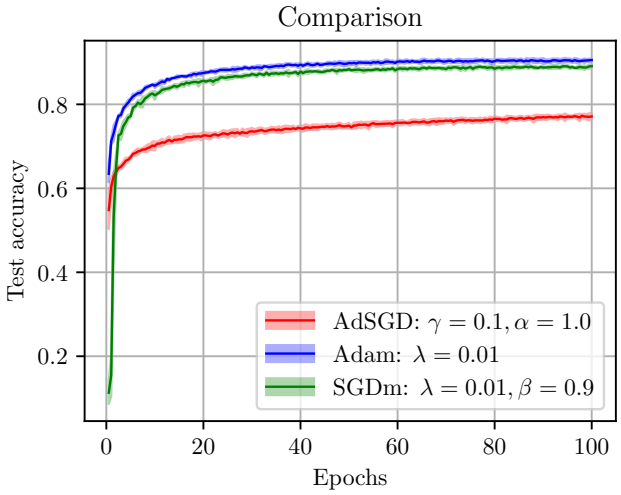


Fig. A.6. Training loss (top), learning rates (middle) and test accuracy (bottom) vs number of epochs for the experiments with the AdSGD, Adam and SGDm optimizers on the FMNIST dataset.

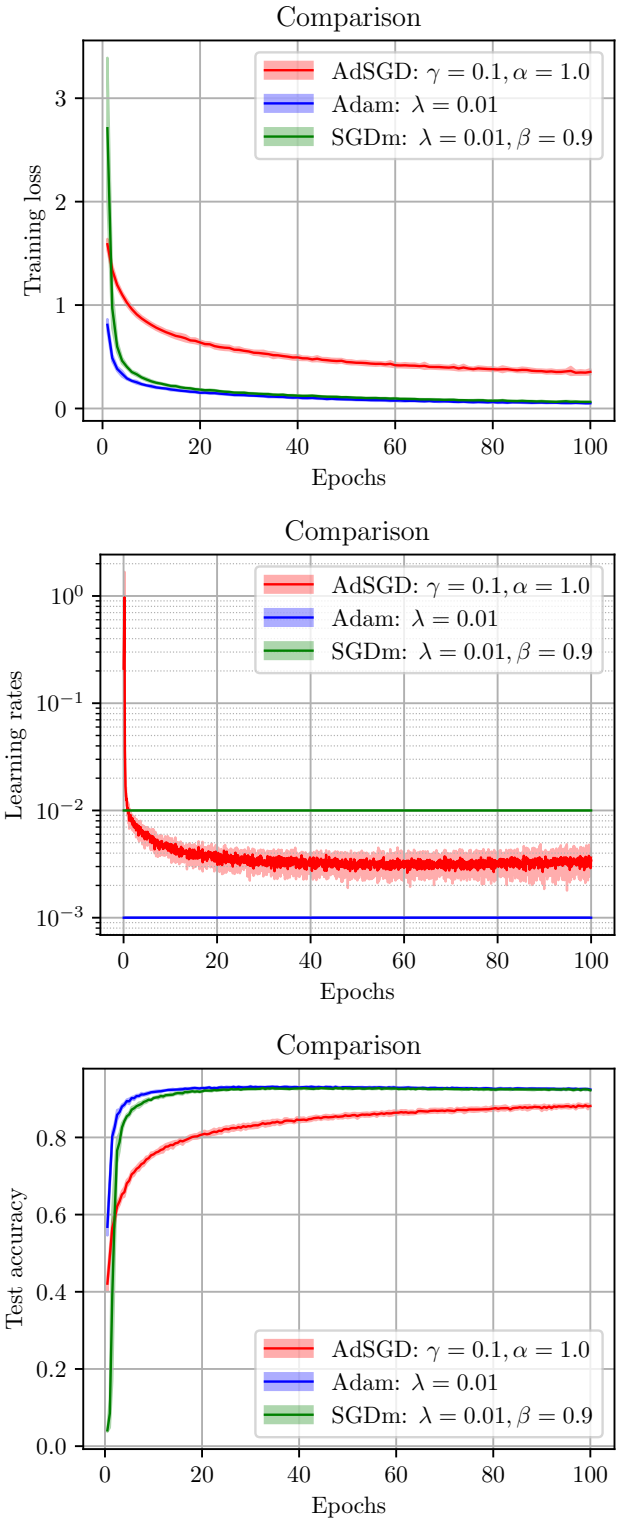


Fig. A.7. Training loss (top), learning rates (middle) and test accuracy (bottom) vs number of epochs for the experiments with the AdSGD, Adam and SGDm optimizers on the EMNIST dataset.

Adaptative Gradient Descent

Under the assumptions that f is convex and ∇f is L-Lipschitz, we have the following convergence rate:

$$f(x_T) - f(x^*) \leq \frac{L}{2T}\|x_0 - x^*\| \tag{A.1}$$

However, this convergence is not guaranteed when ∇f is locally L-Lipschitz (instead of globally). In the paper, the following algorithm is proposed to deal with local Lipschitzness of ∇f.

Algorithm A.1 Adaptive Gradient Descent

Input: $x^0 \in \mathbb{R}^d, \lambda_0 > 0, \theta_0 = +\infty$
$x^1 = x^0 - \lambda_0 \nabla f(x^0)$
for k = 1,2,... **do**
$\lambda_k = \min\{\sqrt{1+\theta_{k-1}}\lambda_{k-1}, \frac{\|x^k - x^{k-1}\|}{2\|\nabla f(x^k) - \nabla f(x^{k-1})\|}\}$
$x^{k+1} = x^k - \lambda_k \nabla f(x^k)$
$\theta_k = \frac{\lambda_k}{\lambda_{k-1}}$
end for