Stochastic Gradient Descent

*Project 2 on the Course “Algorithms for Big Data Analysis”.

1st Chen Yihang

Peking University

1700010780

Abstract

In this report, AdaGrad, ADAM, as well as SGD with BB stepsizes are implemented and tested on MNIST and Covertype (binary) datasets. We found that generally, among three optimization method, i.e. AdaGrad, ADAM, SGD-BB, ADAM is the best choice, and the test accuracy can be up to 90% on MNIST and 75% on Covertype (binary). AdaGrad cannot be on a par with ADAM and SGD-BB, and SGD-BB requires more gradient evaluations to achieve similar performance with ADAM as well as being less stable.

We plot the objective function value, infinity norm of gradient and training 0-1 loss on the training trajectory, and perform grid searches to find the optimal batch size and ℓ_1 penalty parameter. As for the parameters for the optimization process, we directly use their default settings in Pytorch, or in the reference literature.

CONTENTS

I. Introduction

Broadly, this report focuses on the nonsmooth optimization problem

$$\min_{w\in\mathbb{R}^d} \; R(w) := F(w) + \frac{\lambda}{d}\|w\|_1. \tag{I.1}$$

where $F(w) = \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y^i w^T x^i))$ and $\lambda > 0$. I use AdaGrad, ADAM and SGD with BB stepsize to solve the problem. Note that we replace λ with λ/d, where d is the dimension of each vector in the dataset. Such modification can reduce λ's dependence on the datasets.

II. Optimization Algorithms

A. Iterative Soft-Thresholding Algorithm (ISTA)

In the context of solving the ℓ_1-norm regularized problem I.1, the proximal gradient method is

$$w_{k+1} \leftarrow \arg\min_{w\in\mathbb{R}^d}\left(F(w_k) + \nabla F(w_k)^T(w-w_k) + \frac{1}{2\alpha_k}\|w-w_k\|_2^2 + \frac{\lambda}{n}\|w\|_1\right). \tag{II.1}$$

The optimization problem on the right-hand side of this expression is separable and can be solved in closed form. The solution can be written component-wise as

$$[w_{k+1}]_i \leftarrow \begin{cases} [w_k - \alpha_k\nabla F(w_k)]_i + \alpha_k\frac{\lambda}{n} & \text{if } [w_k - \alpha_k\nabla F(w_k)]_i < -\alpha_k\frac{\lambda}{n} \\ 0 & \text{if } [w_k - \alpha_k\nabla F(w_k)]_i \in [-\alpha_k\frac{\lambda}{n}, \alpha_k\frac{\lambda}{n}] \\ [w_k - \alpha_k\nabla F(w_k)]_i - \alpha_k\frac{\lambda}{n} & \text{if } [w_k - \alpha_k\nabla F(w_k)]_i > \alpha_k\frac{\lambda}{n}. \end{cases} \tag{II.2}$$

One also finds that this iteration can be written, with $(\cdot)_+ := \max\{\cdot, 0\}$, as

$$w_{k+1} \leftarrow \mathcal{T}_{\alpha_k\lambda/n}(w_k - \alpha_k\nabla F(w_k)), \quad \text{where} \quad [\mathcal{T}_{\alpha_k\lambda/n}(\tilde{w})]_i = (|\tilde{w}_i| - \alpha_k\lambda/n)_+\operatorname{sgn}(\tilde{w}_i). \tag{II.3}$$

In this form, $\mathcal{T}_{\alpha_k\lambda/n}$ is referred to as the soft-thresholding operator, which leads to the name *iterative soft-thresholding algorithm* (ISTA) being used for (II.1)–(II.2).

B. ADAM

ADAM was proposed in [Kingma and Ba, 2014] and I state it in Algorithm 1. Good default settings are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The algorithm updates exponential moving averages of the gradient (m_t) and the squared gradient (v_t) where

Algorithm 1: Adam

```
Paramters:
        $f$: Stochastic objective function with parameters $\theta$
        $\alpha$: stepsize.
        $\beta_1, \beta_2 \in [0, 1]$: exponential decay rates for the moment estimates.
        $\theta_0$: : initial parameter vector
Return :
        $\theta_t$: resulting parameters
1  $m_0 \leftarrow 0$ (Initialize 1st moment vector);
2  $v_0 \leftarrow 0$ (Initialize 2nd moment vector);
3  $t \leftarrow 0$ (Initialize timestep);
4  while not converged do
5      $t \leftarrow t + 1$;
6      $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$);
7      $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ (Update biased first moment estimate);
8      $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ (Update biased second raw moment estimate);
9      $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate);
10     $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate);
11     $\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters);
12 end
```

the hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages. The moving averages themselves are estimates of the 1st moment (the mean) and the 2nd raw moment (the uncentered variance) of the gradient.

C. *AdaGrad*

In essence, AdaGrad [Duchi et al., 2011] use $H_k = \frac{1}{\alpha}\text{diag}(\sum_{i=1}^{k} g_i^2)^{1/2}$ to approximate Hessian matrix in Newton method, which leads to the update rule

$$x_{k+1} = x_k - H_k^{-1} g_k \tag{II.4}$$

where we add a constant ϵ to prevent zero-division error.

D. *SGD-BB*

- Barzilai-Borwein stepsize

  The BB method, proposed by Barzilai and Borwein in [Barzilai and Borwein, 1988], has been proved to be very successful in solving nonlinear optimization problems.

Algorithm 2: AdaGrad

```
Paramters:  f: Stochastic objective function with parameters θ
            α: stepsize.
            ϵ: small costant to prevent zero-division.
            θ_0: : initial parameter vector
Return    :
            θ_t: resulting parameters
v_0 ← 0 (Initialize diagnoal matrix);
t ← 0 (Initialize squared sum of gradient);
while not converged do
    t ← t + 1;
    g_t ← ∇_θ f_t(θ_{t−1}) (Get gradients w.r.t. stochastic objective at timestep t);
    v_t ← v_{t−1} + g_t^2 (Update squared sum of gradient);
    α_t = α/(√v_t + ϵ) (Update stepsize);
    θ_t = θ_{t−1} − α_t g_t) (Update parameters);
end
```

The key idea behind the BB method is motivated by quasi-Newton methods. Suppose we want to solve the unconstrained minimization problem

$$\min_x \; f(x), \tag{II.5}$$

where f is differentiable. A typical iteration of quasi-Newton methods for solving (II.5) takes the following form:

$$x_{t+1} = x_t - B_t^{-1} \nabla f(x_t), \tag{II.6}$$

where B_t is an approximation of the Hessian matrix of f at the current iterate x_t. Different choices of B_t give different quasi-Newton methods. The most important feature of B_t is that it must satisfy the so-called secant equation:

$$B_t s_t = y_t, \tag{II.7}$$

where $s_t = x_t - x_{t-1}$ and $y_t = \nabla f(x_t) - \nabla f(x_{t-1})$ for $t \geq 1$.

It is noted that in (II.6) one needs to solve a linear system, which may be time consuming when B_t is large and dense.

One way to alleviate this burden is to use the BB method, which replaces B_t by a scalar matrix $\frac{1}{\eta_t} I$. However, one cannot choose a scalar η_t such that the secant equation (II.7) holds with $B_t = \frac{1}{\eta_t} I$. Instead, one can find η_t such that the residual of the secant equation is minimized, i.e.,

$$\min_{\eta_t} \left\| \frac{1}{\eta_t} s_t - y_t \right\|_2^2,$$

which leads to the following choice of η_t:

$$\eta_t = \frac{\|s_t\|_2^2}{s_t^T y_t}. \tag{II.8}$$

Therefore, a typical iteration of the BB method for solving (II.5) is

$$x_{t+1} = x_t - \eta_t \nabla f(x_t), \tag{II.9}$$

where η_t is computed by (II.8).

The BB method does not apply to SGD directly, because SGD never computes the full gradient $\nabla F(x)$. In SGD, $\nabla f_{i_t}(x_t)$ is an unbiased estimation for $\nabla F(x_t)$ when i_t is uniformly sampled. Therefore, one may suggest to use $\nabla f_{i_{t+1}}(x_{t+1}) - \nabla f_{i_t}(x_t)$ to estimate $\nabla F(x_{t+1}) - \nabla F(x_t)$ when computing the BB step size using formula (II.8). However, this approach does not work well because of the variance of the stochastic gradient estimates. [Tan et al., 2016] proposes to incorporate the BB method to SGD in Algorithm 3

III. Numerical Results

When implementing the stochastic gradient method, we regard "training set size"/"batch size" iterations as one epoch. Hence the computational budget of one epoch amounts to one full gradient. We are able to compare different method given the number of same gradient evaluations, which is usually larger than what is required to be converged. The output values on each epoch are averaged. We use 30 epoch in the training phase for MNIST and 20 for Covertype.

The codes can be readily modified to set some stopping condition and terminate the training before the maximum number of iteration is achieved. However, since we want to compare their performace rather to train a classifier, we do not perform experiments on this subject.

A. Files description

1) .\AdaGrad.m: implements one step of AdaGrad.

Algorithm 3: SGD with BB step size (SGD-BB)

```
Paramters:
        $m$: update frequency.
        $f$: Stochastic objective function with parameters $\theta$
        $\tilde{x}_0$: initial point.
        $\eta_0$, $\eta_1$: initial step size (only used in the first epoch)
Return :
        $x_k$: resulting parameters
for $k = 0, 1, \cdots$ do
    if $k > 0$ then
        $\eta_k = \frac{1}{m} \cdot \|\tilde{x}_k - \tilde{x}_{k-1}\|_2^2 / |(\tilde{x}_k - \tilde{x}_{k-1})^T(\hat{g}_k - \hat{g}_{k-1})|$;
    end
    $x_0 = \tilde{x}_k$;
    $\hat{g}_{k+1} = 0$;
    for $t = 0, \cdots, m-1$ do
        Randomly pick $i_t \in \{1, \dots, n\}$;
        $x_{t+1} = x_t - \eta_k \nabla f_{i_t}(x_t)$                    $(*)$;
        $\hat{g}_{k+1} = \beta \nabla f_{i_t}(x_t) + (1-\beta)\hat{g}_{k+1}$;
    end
    $\tilde{x}_{k+1} = x_m$
end
```

2) .**ADAM.m**: implements one step of ADAM.
3) .**SGD-BB.m**: implements the full process of SGD-BB.
4) .**objective.m**: implements the objective function.
5) .**mnist_loader.m**: load the MNIST dataset.
6) .**mnist_test.m**: test the three optimization methods on MNIST given the batchsize and λ.
7) .**covertype_loader.m**: load the Covertype.binary dataset.
8) .**covertype_test.m**: test the three optimization methods on Covertype.binary given the batchsize and λ.
9) .**test1.m**: generates figures and tables in Section III-B.
10) .**test2.m**: generates figures and tables in Section III-C.
11) .**data****covertype**: Covertype.binary dataset.
12) .**data****mnist**: MNIST dataset.

B. MNIST

The MNIST database [LeCun et al., 1998] of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

The data in MNIST are 28*28 images and labels are integers from 0 to 9. We modify the label to be -1 and 1 so that even numbers become 1 and odd become -1. The training set and test set have been divided. In the training set, we reshape the data into a 784×60000 matrix, whose item is an integer ranging from 0 to 1. (Here we do resize the original item [0,255] into [0,1]).

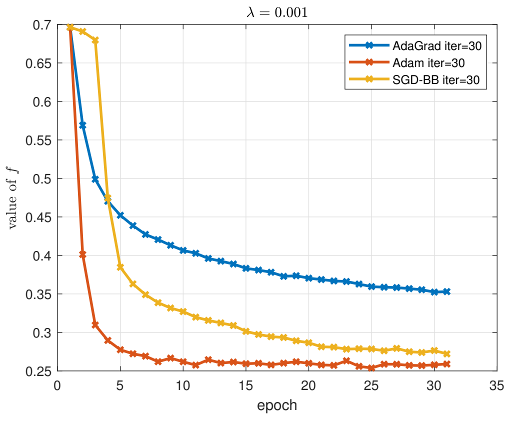
The MNIST dataset can be directly retrieved from The MNIST DATABASE of handwritten digits by "mnist_loader.m".

We set the batchsize to be $200, 1000, 5000, 10000$. For each step, we sample a different batch to evaluate the gradient. In the end, we use the output w to calculate the 0-1 loss on the test data.

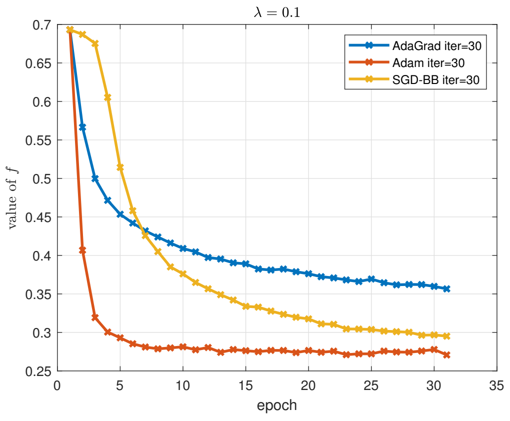
batchsize	λ	0.001	0.1	1	10
	AdaGrad	0.8620	0.8621	0.8515	0.8320
200	Adam	0.8958	0.8949	0.8756	0.8362
	SGD-BB	0.8872	0.8926	0.8728	0.8354
	AdaGrad	0.8455	0.8445	0.8404	0.8414
1000	Adam	0.8855	0.8856	0.8710	0.8342
	SGD-BB	0.8862	0.8844	0.8693	0.8341
	AdaGrad	0.8221	0.8216	0.8177	0.8217
5000	Adam	0.8715	0.8717	0.8614	0.8337
	SGD-BB	0.8750	0.8712	0.8678	0.8337
	AdaGrad	0.8127	0.8132	0.8088	0.8122
10000	Adam	0.8590	0.8584	0.8528	0.8338
	SGD-BB	0.8766	0.8711	0.8540	0.8327

Table III.1
TEST ACCURACY ON THE MNIST MODEL

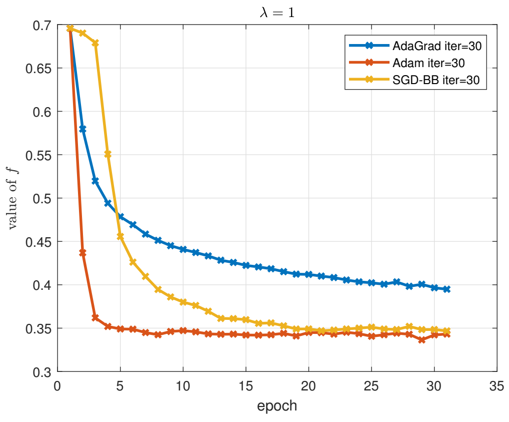
We find that setting batch size to be 200, and $\lambda = 0.01 \sim 0.1$ seems to be the optimal choice. Generally, we find that SGD-BB and ADAM outperform AdaGrad. Besides, ADAM decreases more rapidly in the first few iterations than SGD-BB. Hence, ADAM is the best choice in analyzing MNIST datasets.



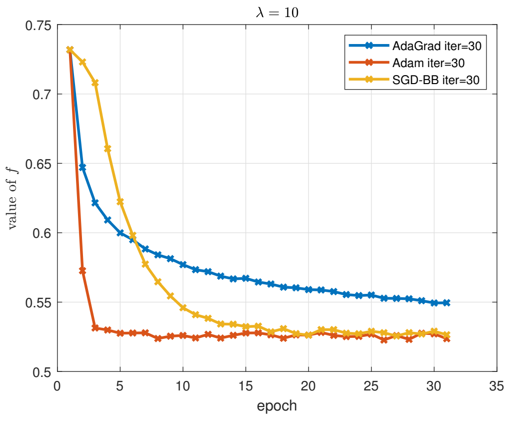
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

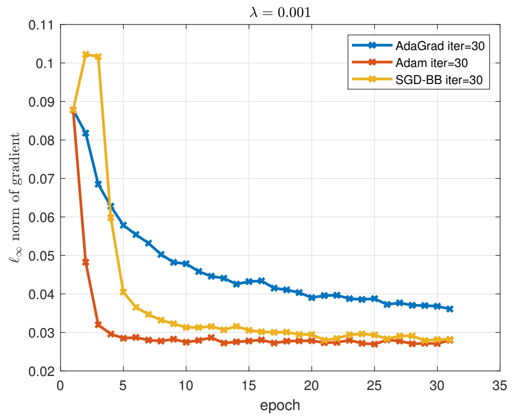


(c) $\lambda = 1$

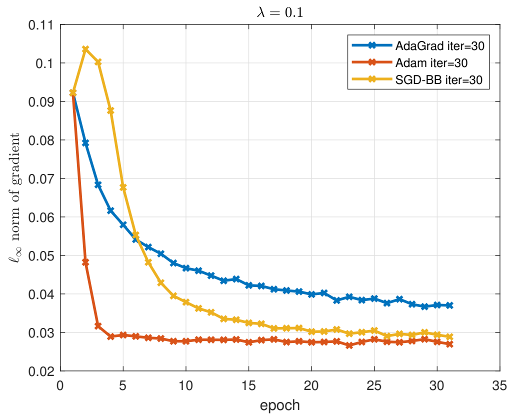


(d) $\lambda = 10$

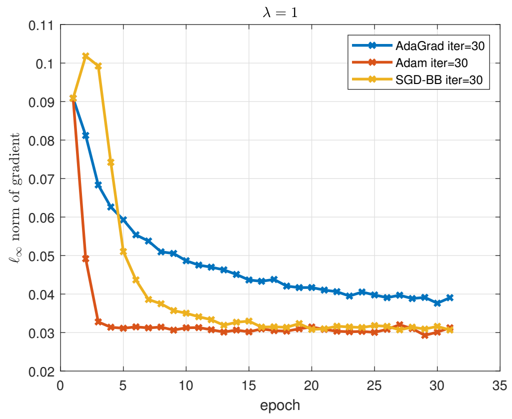
Figure III.1. Surrogate Loss on the MNIST model, batch size = 200



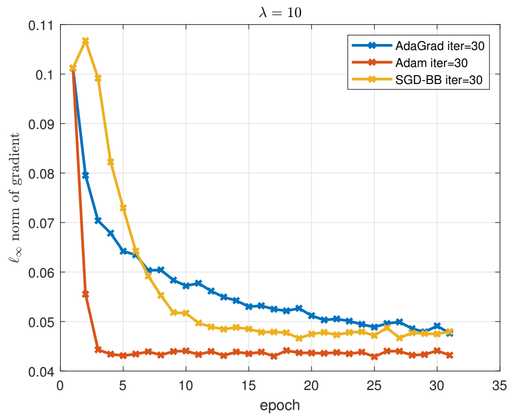
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

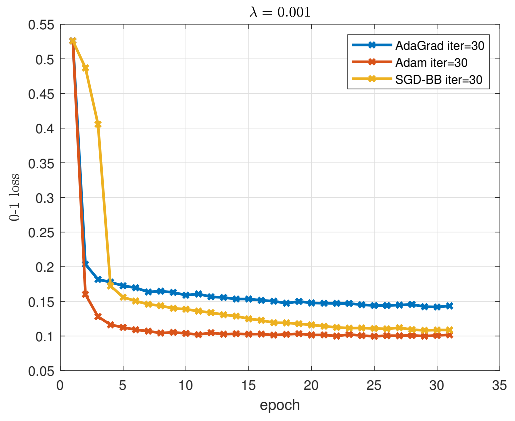


(c) $\lambda = 1$

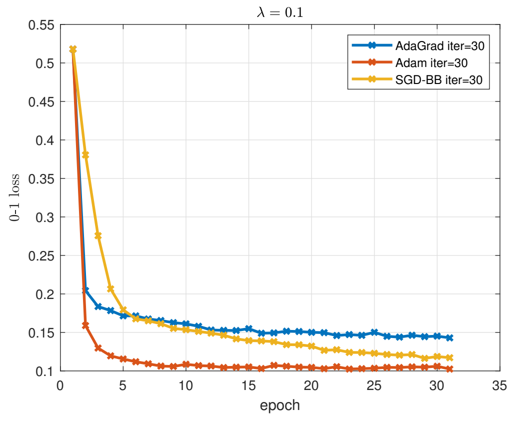


(d) $\lambda = 10$

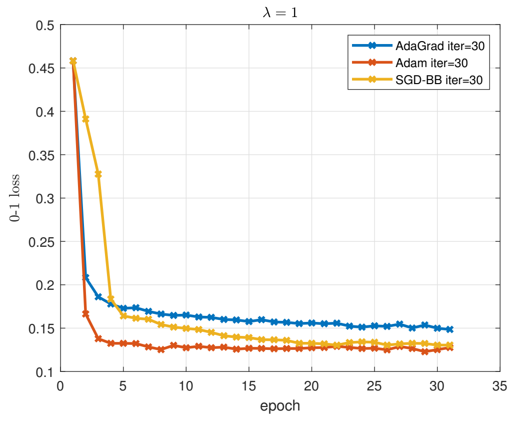
Figure III.2. Infinity norm of gradient on the MNIST model, batch size = 200



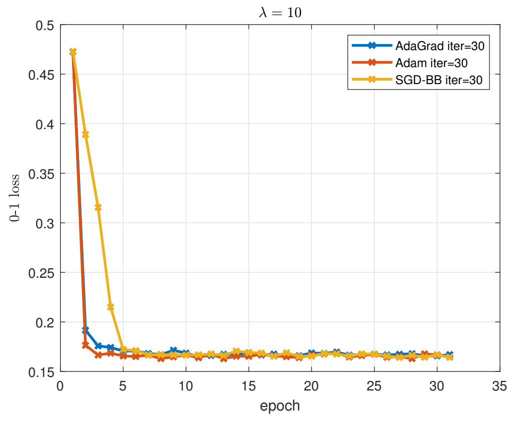
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

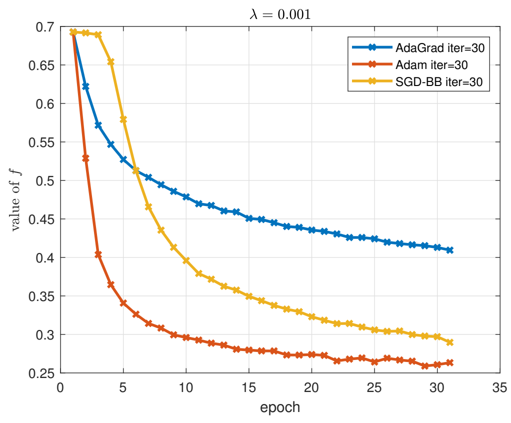


(c) $\lambda = 1$

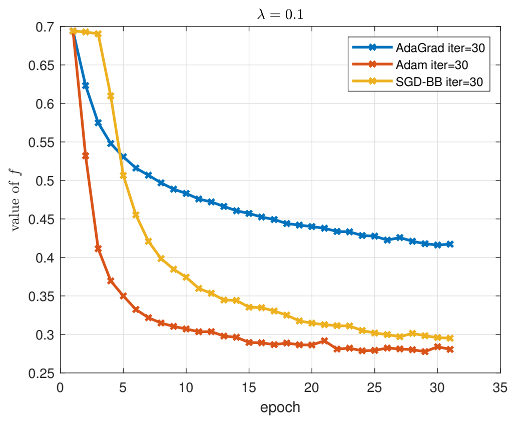


(d) $\lambda = 10$

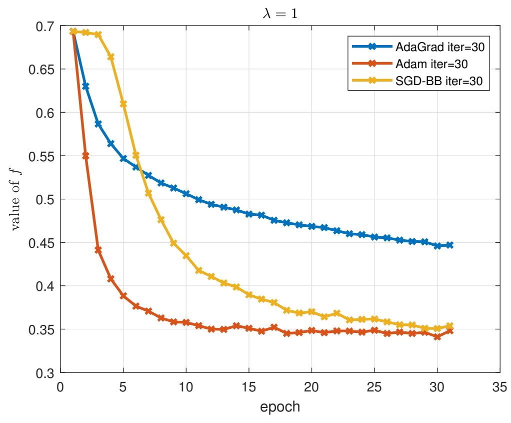
Figure III.3. 0-1 Loss on the MNIST model, batch size = 200



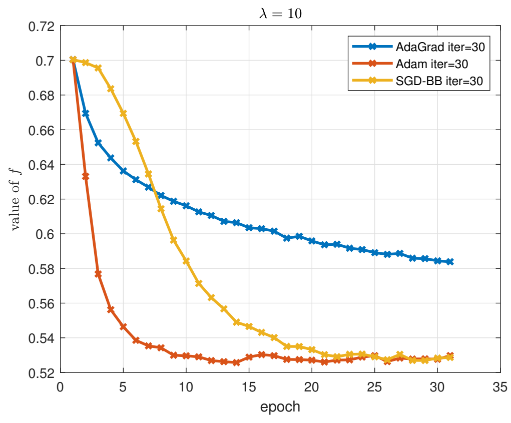
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

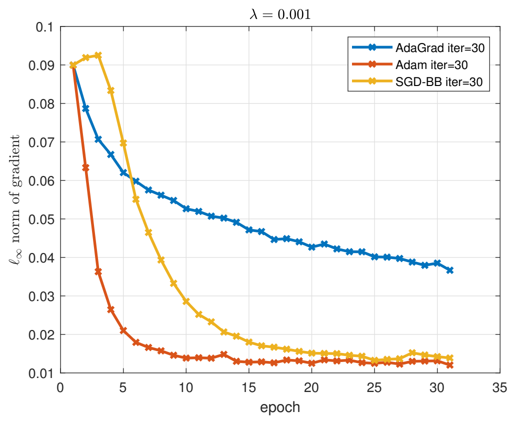


(c) $\lambda = 1$

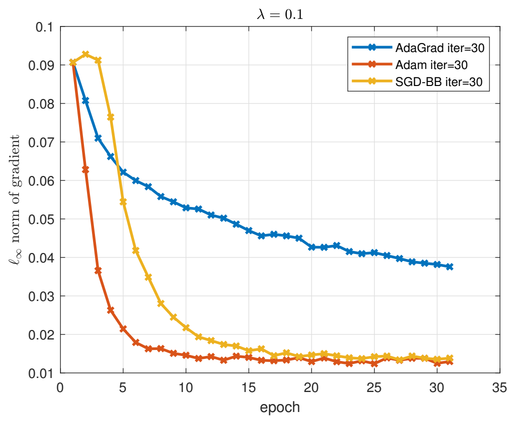


(d) $\lambda = 10$

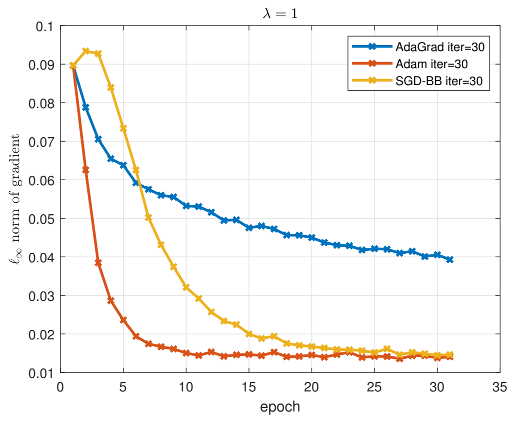
Figure III.4. Surrogate Loss on the MNIST model, batch size = 1000



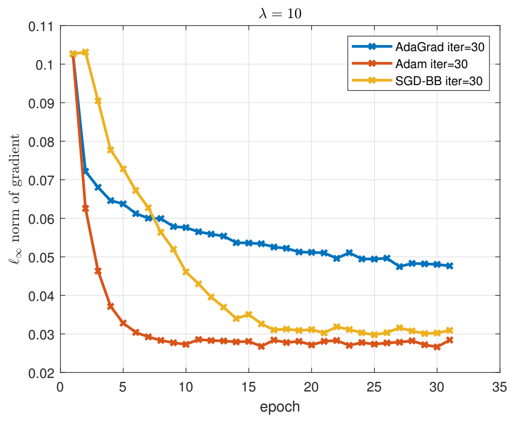
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

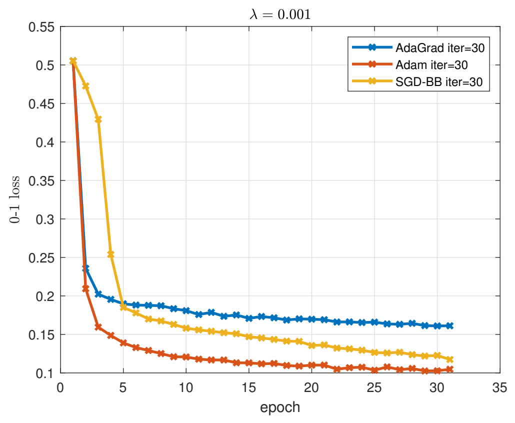


(c) $\lambda = 1$

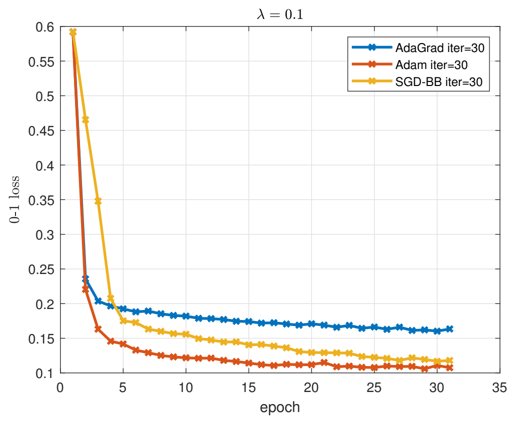


(d) $\lambda = 10$

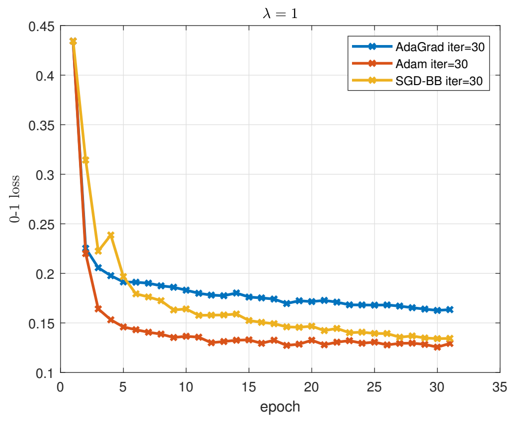
Figure III.5. Infinity norm of gradient on the MNIST model, batch size = 1000



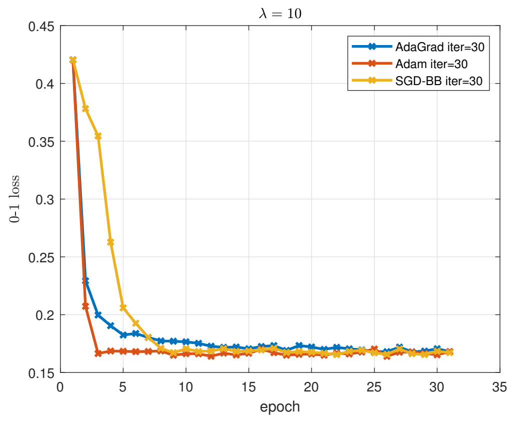
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

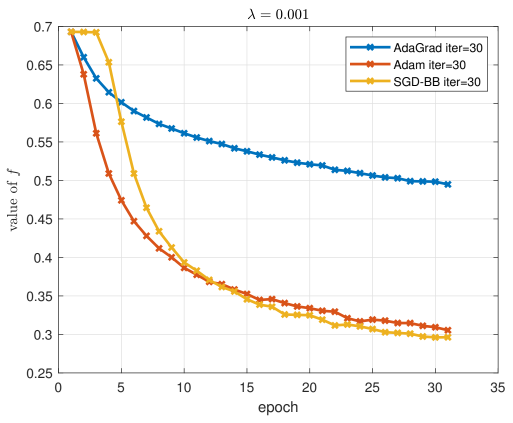


(c) $\lambda = 1$

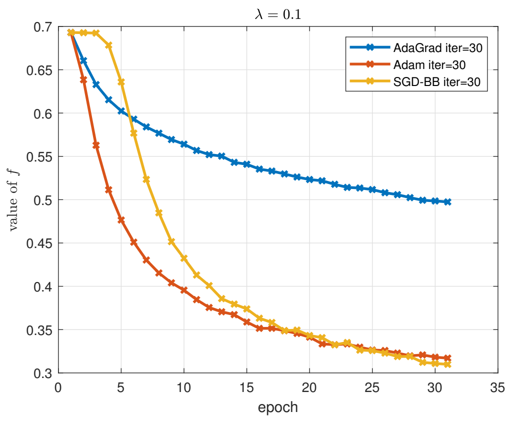


(d) $\lambda = 10$

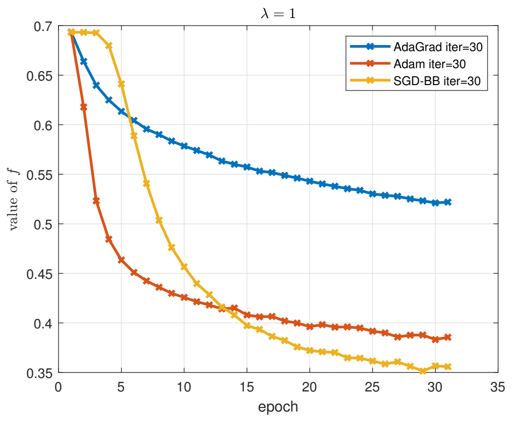
Figure III.6. 0-1 Loss on the MNIST model, batch size = 1000



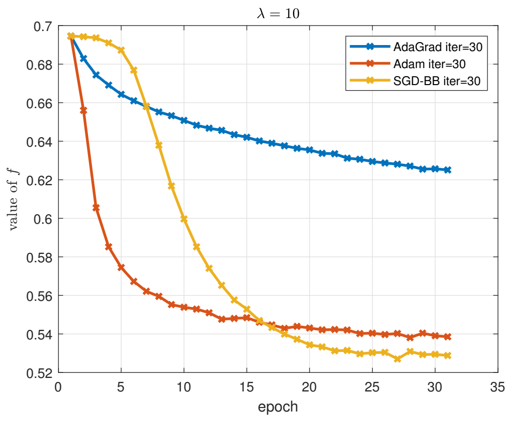
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

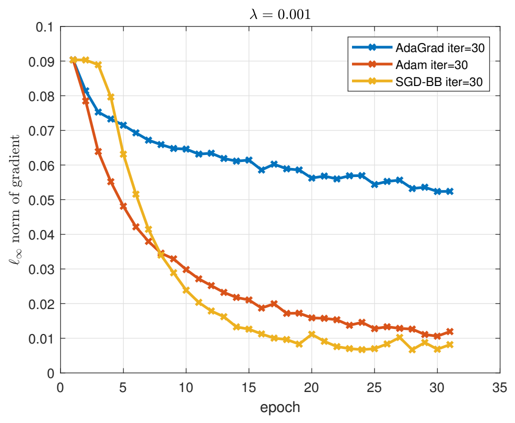


(c) $\lambda = 1$

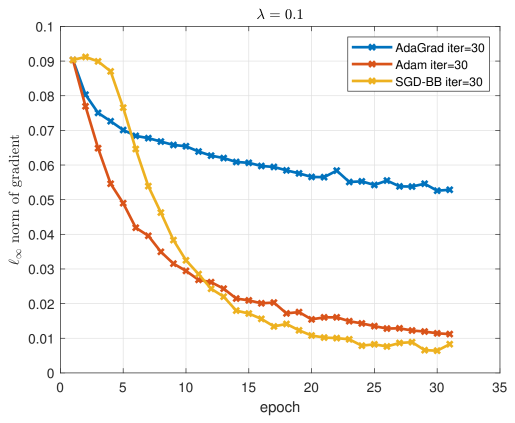


(d) $\lambda = 10$

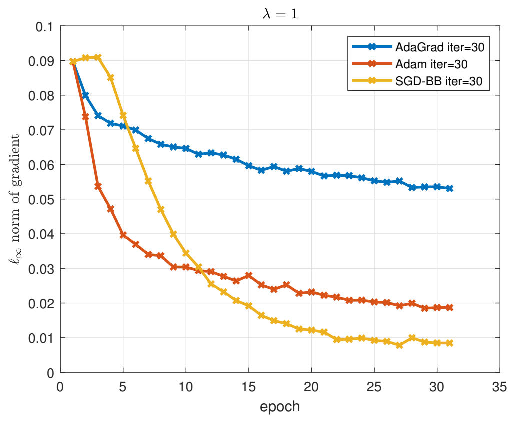
Figure III.7. Surrogate Loss on the MNIST model, batch size = 5000



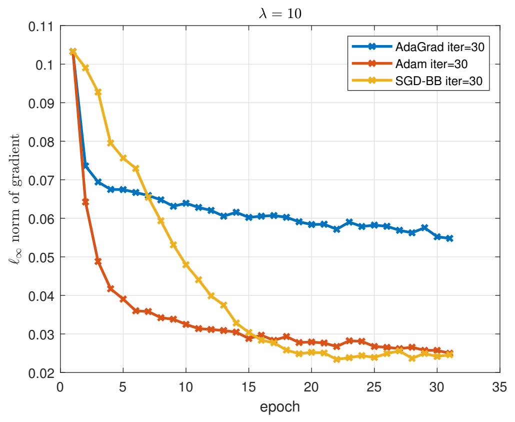
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

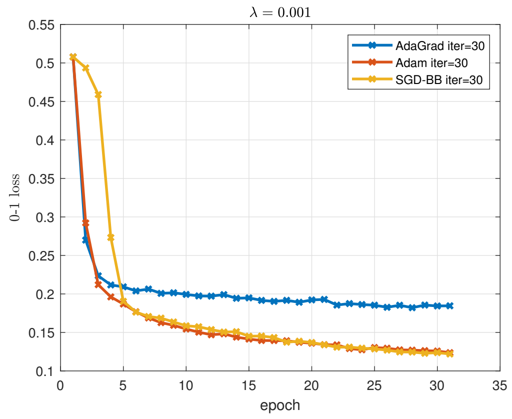


(c) $\lambda = 1$

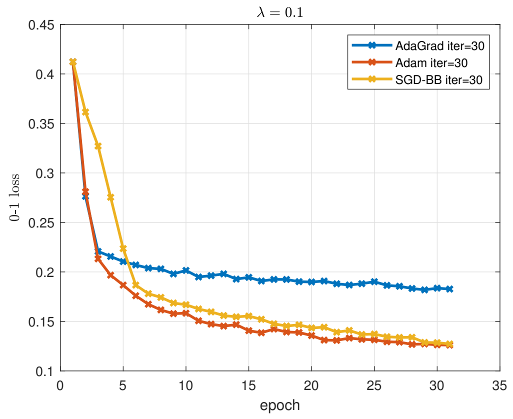


(d) $\lambda = 10$

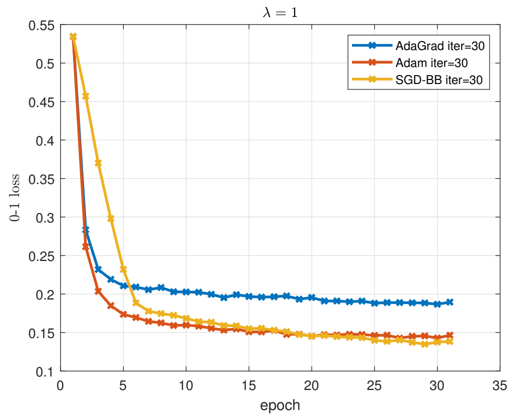
Figure III.8. Infinity norm of gradient on the MNIST model, batch size = 5000



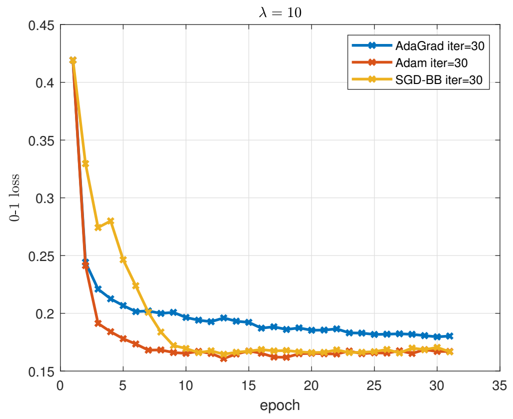
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

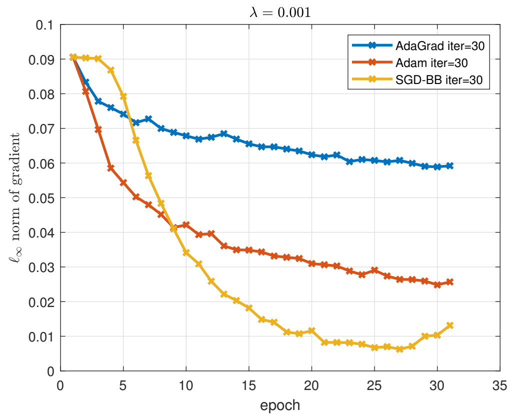


(c) $\lambda = 1$

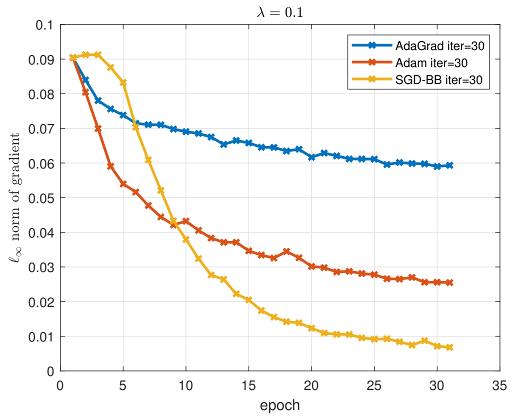


(d) $\lambda = 10$

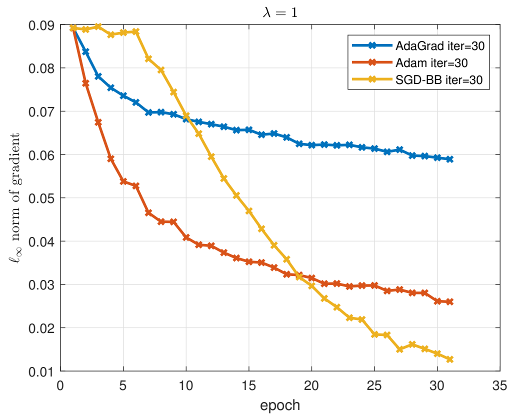
Figure III.9. 0-1 Loss on the MNIST model, batch size = 5000



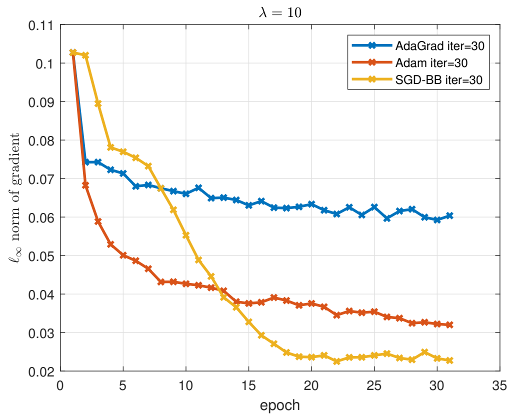
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

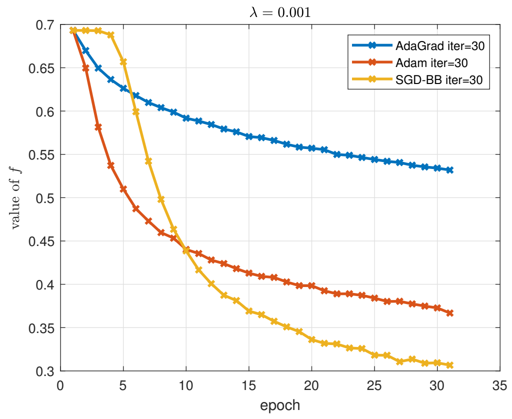


(c) $\lambda = 1$

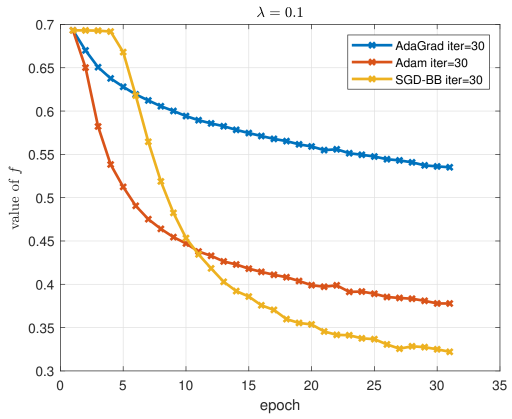


(d) $\lambda = 10$

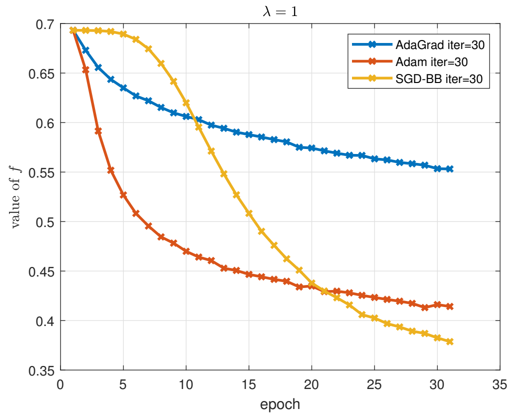
Figure III.10. Infinity norm of gradient on the MNIST model, batch size = 10000



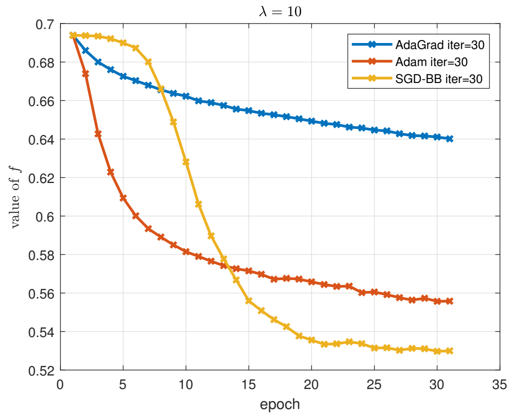
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$

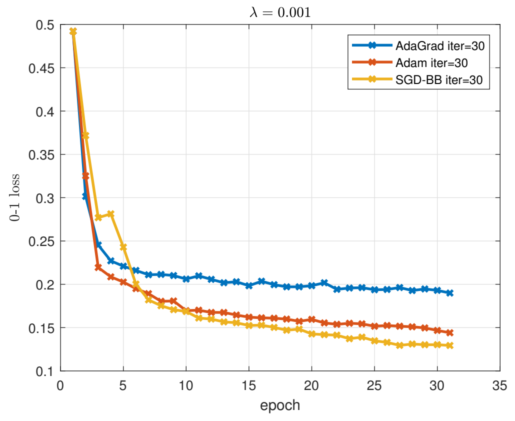


(c) $\lambda = 1$

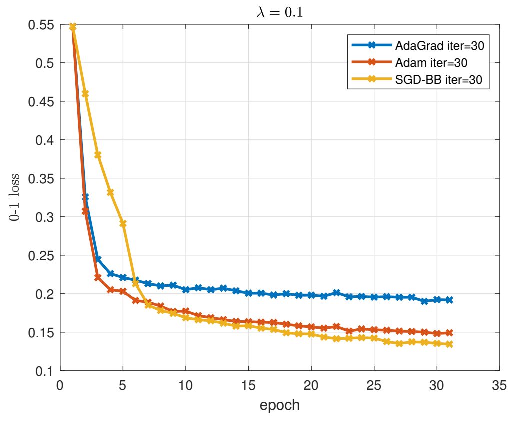


(d) $\lambda = 10$

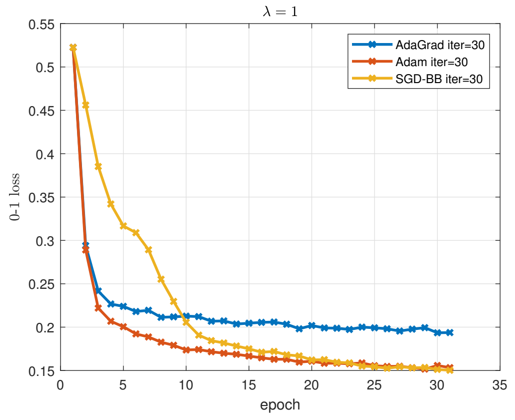
Figure III.11. Surrogate Loss on the MNIST model, batch size = 10000



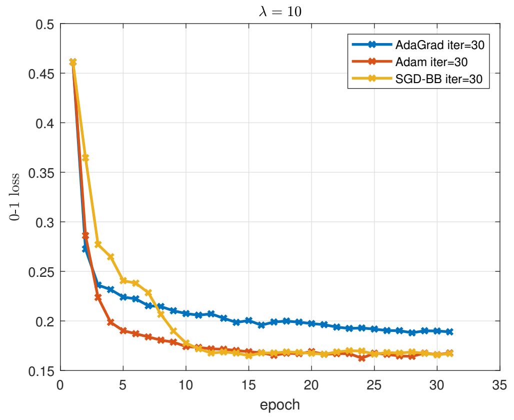
(a) $\lambda = 0.001$



(b) $\lambda = 0.1$



(c) $\lambda = 1$



(d) $\lambda = 10$

Figure III.12. 0-1 Loss on the MNIST model, batch size = 10000

C. Covertype

The actual forest cover type for a given observation (30 x 30 meter cell) was determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. [Blackard, 2000] Independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data. Data is in raw form (not scaled) and contains binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types).[1]

However, the original dataset does not contain binary data. We use covertype.binary instead. In the following, we detail how to install the LIBSVM package on Windows.

We first download package from LIBSVM, then install MATLAB Support for MinGW-w64 C/C++ Compiler. Before executing "make.m" on Windows 10, we are required to replace "CFLAGS" by "COMPFLAGS" (suggested by this link) in it. After that, the LIBSVM can be readily installed. We use "libsvmread" to read load the covertype.binary datasets.

The data in Covertype are 581012*54 matrix, with the label 1 or 2. We still modify the label to be -1 and 1. We choose 450000 observations as training set (around 80%), and select 10000 data with label 1 as well as 10000 data with label -1 as test set. The training data can be resized as 45000×54 matrix. Then, we reshuffle each the rows randomly so that we can retrieve the training dataset and test dataset uniformly.

We set the batchsize to be $500, 1000, 5000, 10000$. For each step, we sample a different batch to evaluate the gradient. In the end, we use the output w to calculate the 0-1 loss on the test data.

We find that we cannot use SGD-BB to training on the raw data, which could be attributed to the zero division error. Hence, we try, for each column, resizing the items in it so that the largest one will be 1. However, we find that such preprocessing will be detrimental to performance. Hence, we decide not to adopt the SGD-BB method in our experiments.

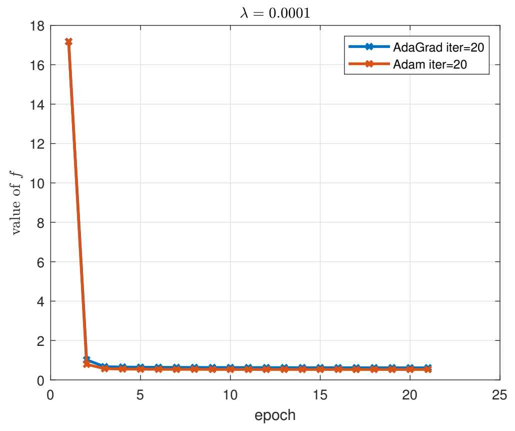
We find that despite the fact that ADAM and AdaGrad is able to achieve similar values of the objective function, their performance is drastically different, which could

[1]http://archive.ics.uci.edu/ml/datasets/covertype

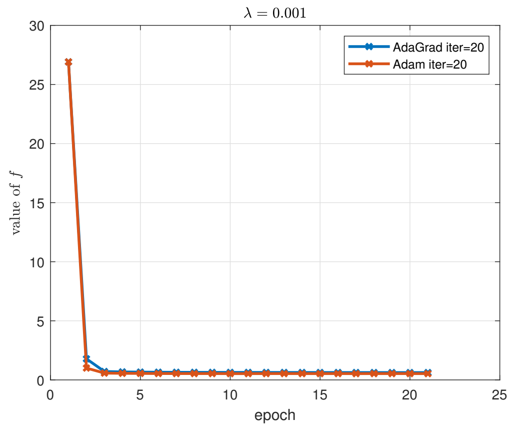
be attributed to the differences in their gradients. In general, we find that optimizing by ADAM, setting the batch size to be 500 and $\lambda = 0.001$ can achieve 75% accuracy.

batchsize	λ	0.0001	0.001	0.01	0.1
	AdaGrad	0.6459	0.6512	0.6535	0.6526
500	Adam	0.7183	0.7424	0.7295	0.7329
	AdaGrad	0.6264	0.6437	0.6419	0.6416
1000	Adam	0.6751	0.7345	0.7309	0.7349
	AdaGrad	0.6272	0.6113	0.4334	0.4491
5000	Adam	0.7128	0.7297	0.7298	0.7319
	AdaGrad	0.5555	0.5440	0.6089	0.6218
10000	Adam	0.6818	0.7090	0.6866	0.6953

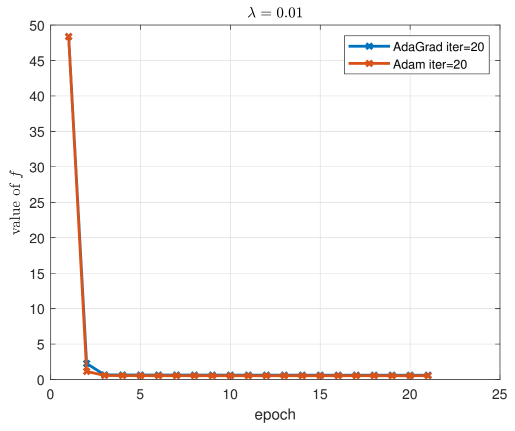
Table III.2
TEST ACCURACY ON THE COVERTYPE MODEL



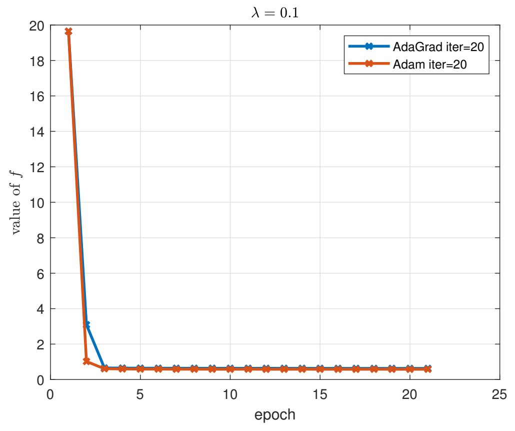
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

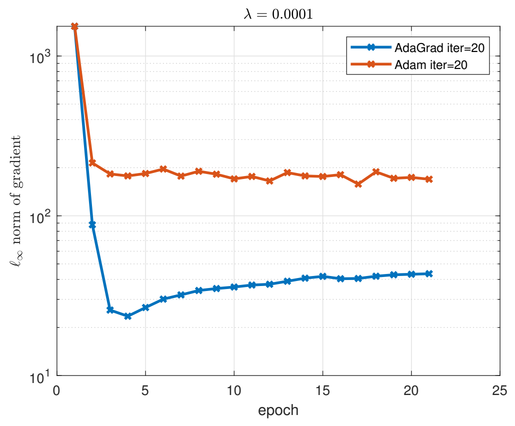


(c) $\lambda = 1$

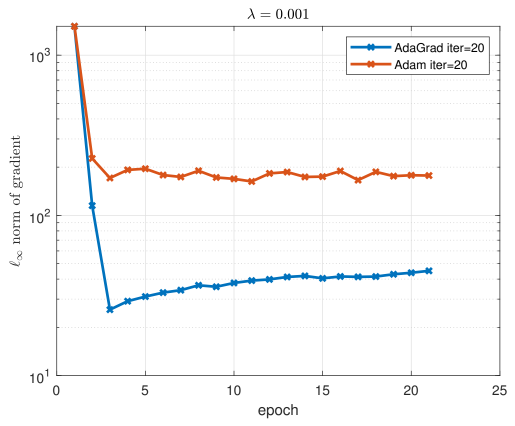


(d) $\lambda = 0.1$

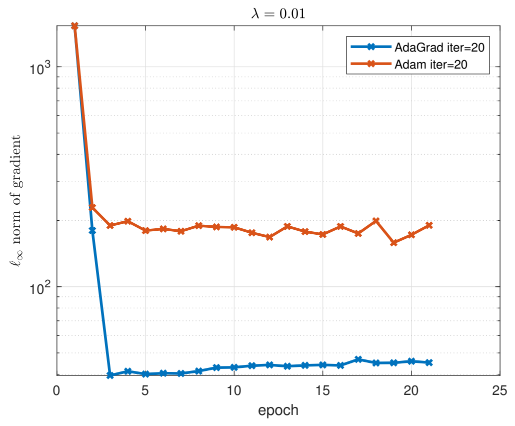
Figure III.13. Surrogate Loss on the MNIST model, batch size = 500



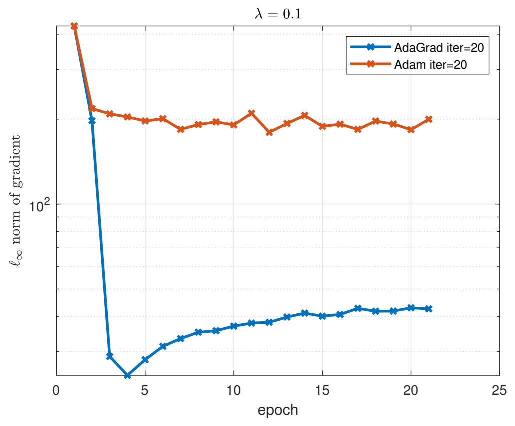
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

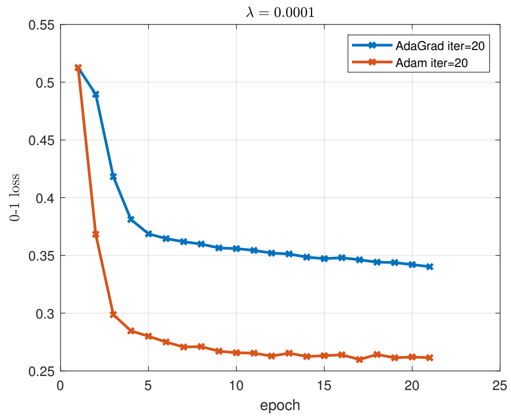


(c) $\lambda = 1$

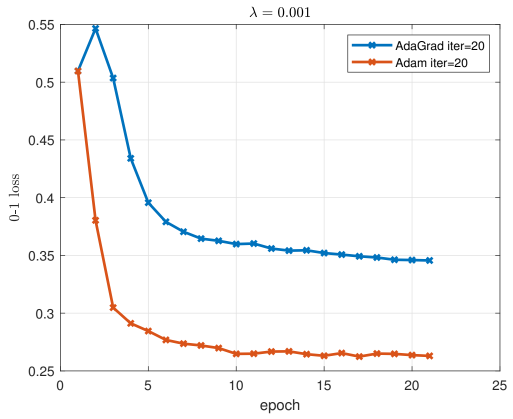


(d) $\lambda = 0.1$

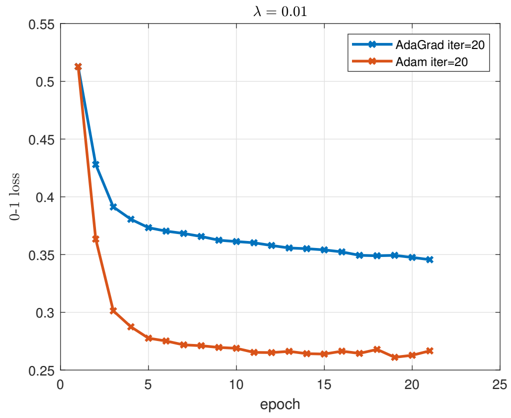
Figure III.14. Infinity norm of gradient on the MNIST model, batch size = 500



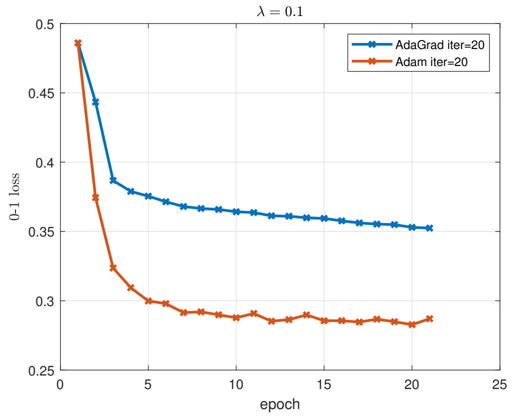
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

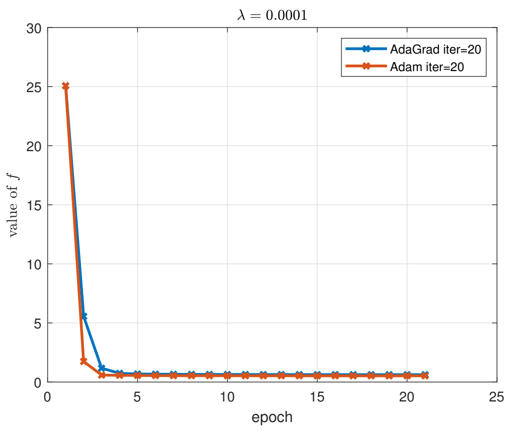


(c) $\lambda = 1$

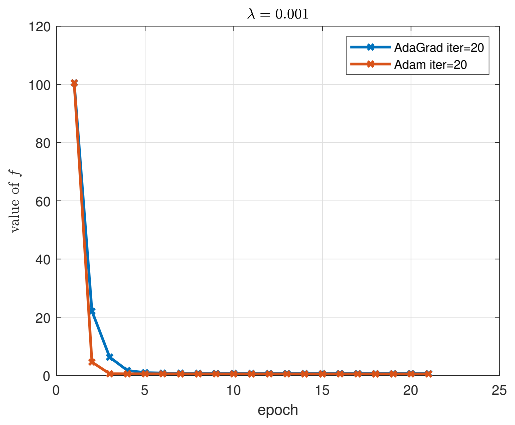


(d) $\lambda = 0.1$

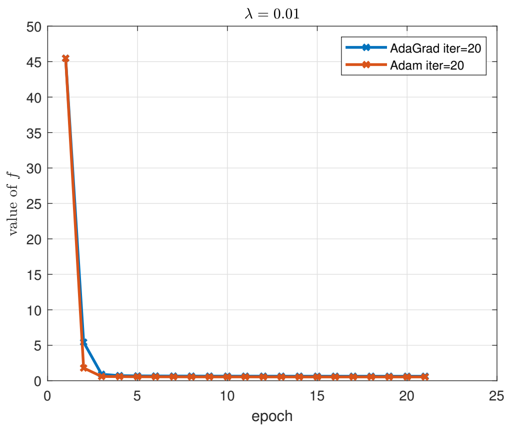
Figure III.15. 0-1 Loss on the MNIST model, batch size = 500



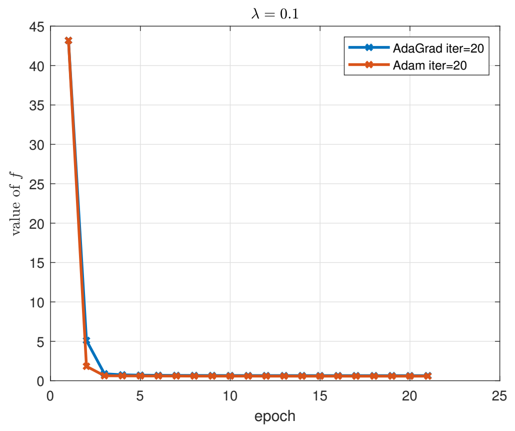
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

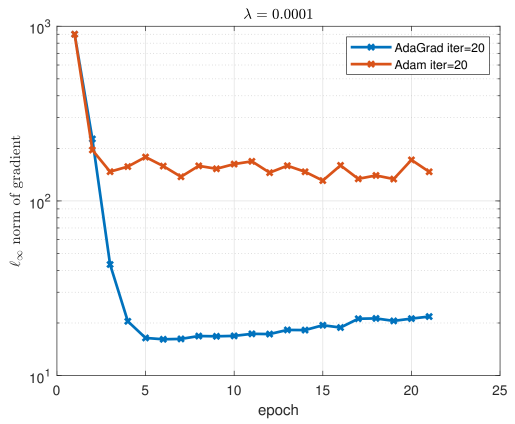


(c) $\lambda = 0.01$

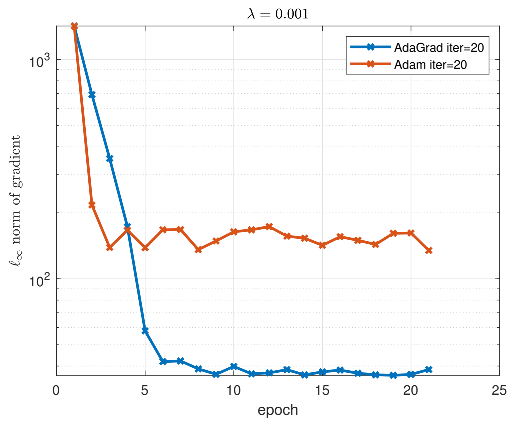


(d) $\lambda = 0.1$

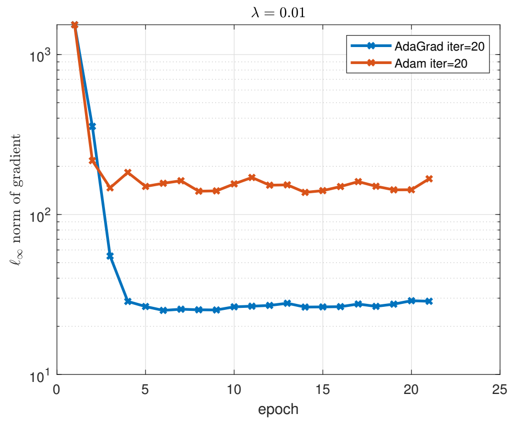
Figure III.16. Surrogate Loss on the MNIST model, batch size = 1000



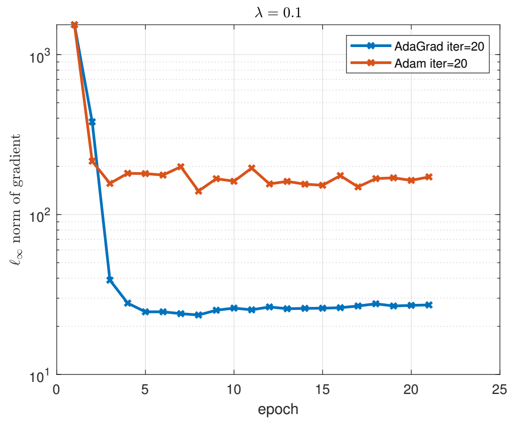
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

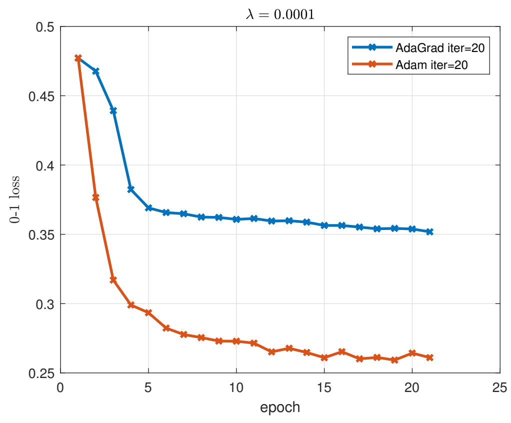


(c) $\lambda = 0.01$

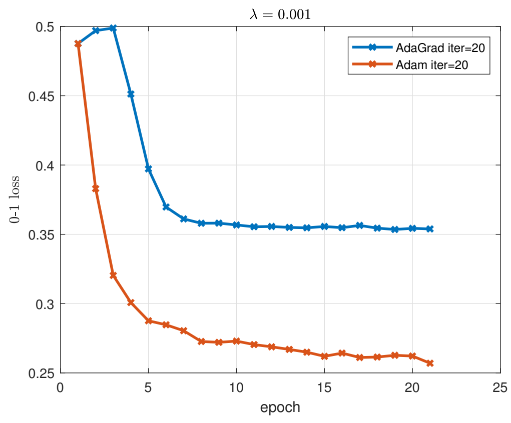


(d) $\lambda = 0.1$

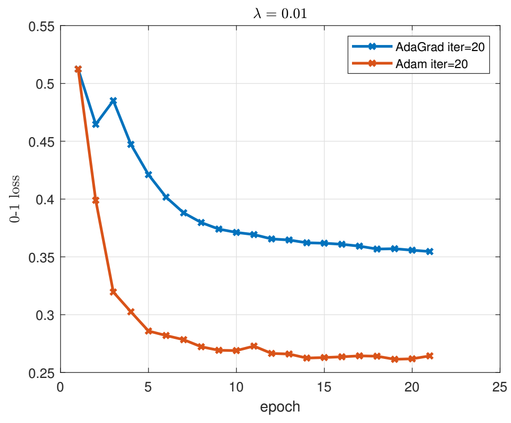
Figure III.17. Infinity norm of gradient on the MNIST model, batch size = 1000



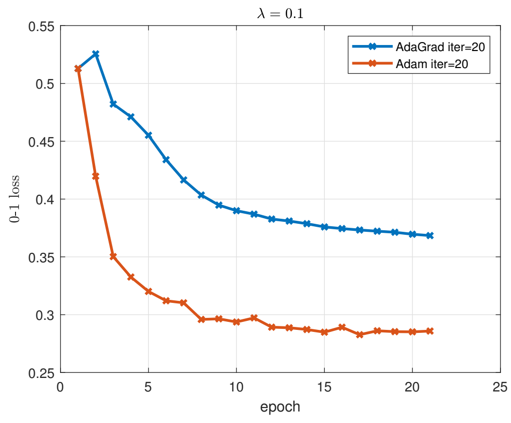
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

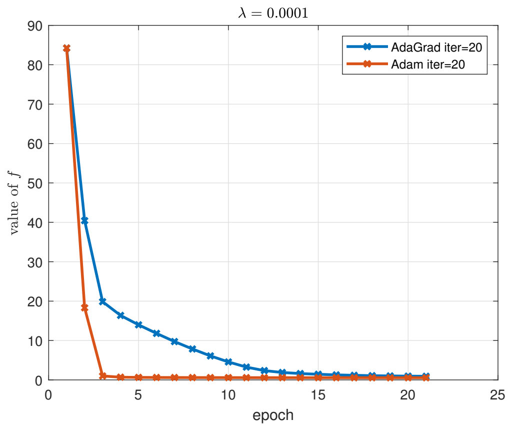


(c) $\lambda = 0.01$

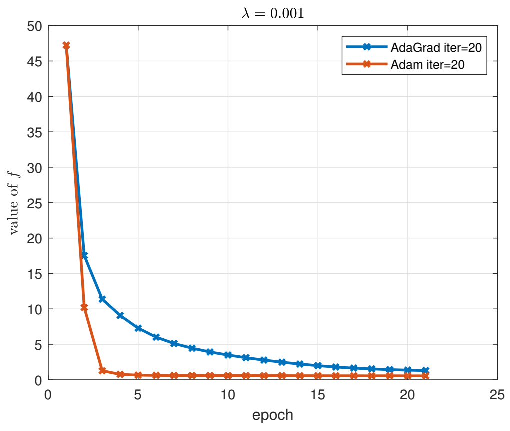


(d) $\lambda = 0.1$

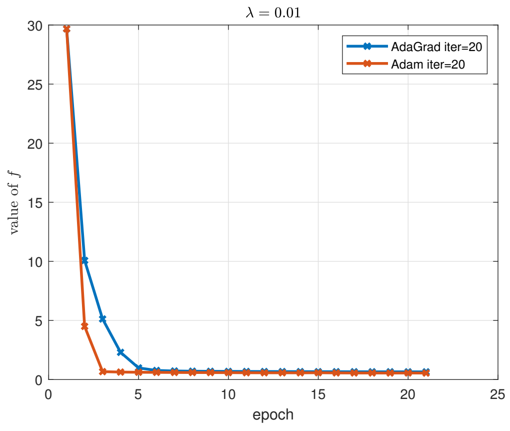
Figure III.18. 0-1 Loss on the MNIST model, batch size = 1000



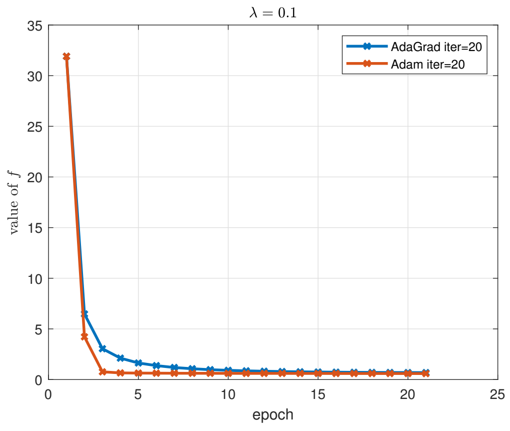
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

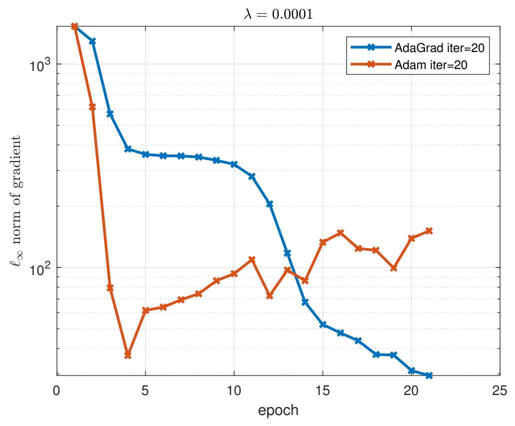


(c) $\lambda = 0.01$

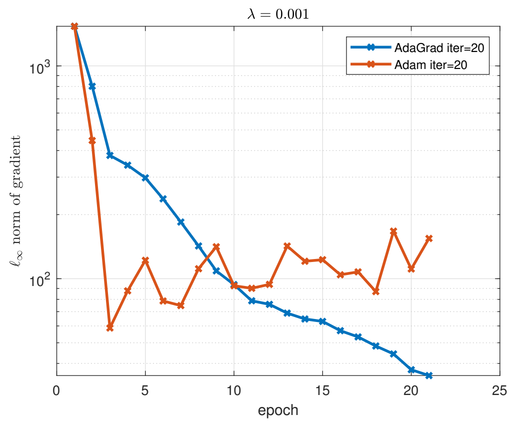


(d) $\lambda = 0.1$

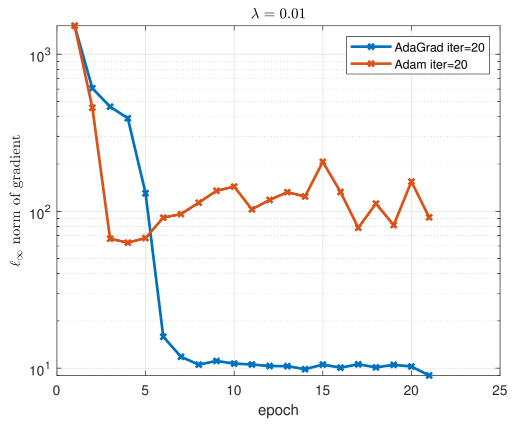
Figure III.19. Surrogate Loss on the MNIST model, batch size = 5000



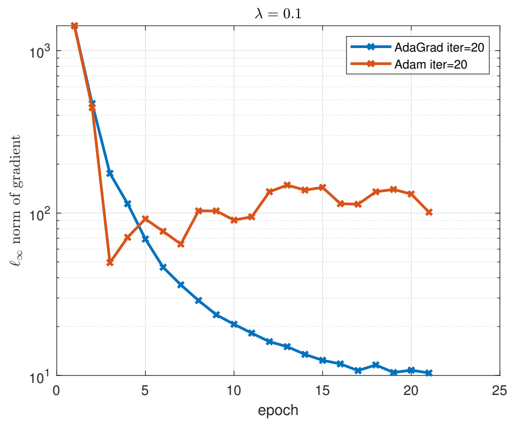
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

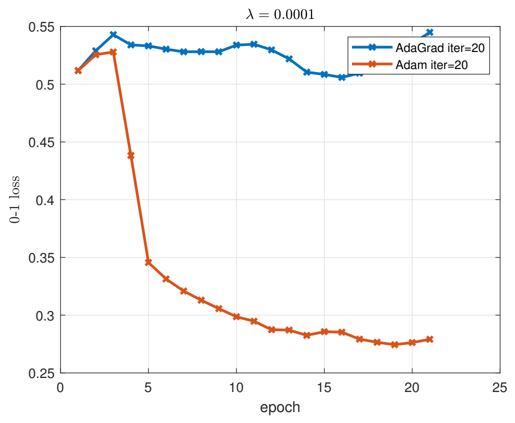


(c) $\lambda = 0.01$

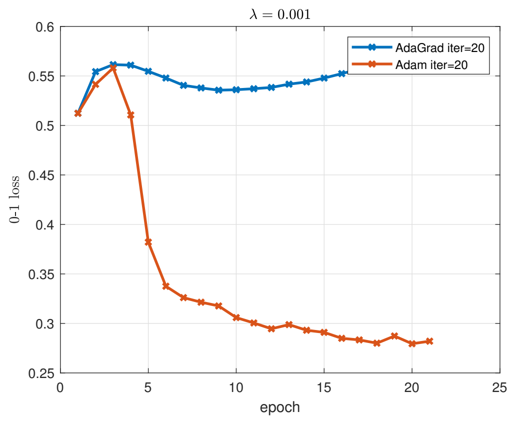


(d) $\lambda = 0.1$

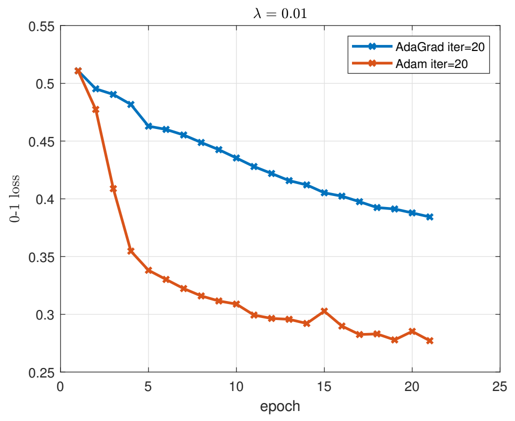
Figure III.20. Infinity norm of gradient on the MNIST model, batch size = 5000



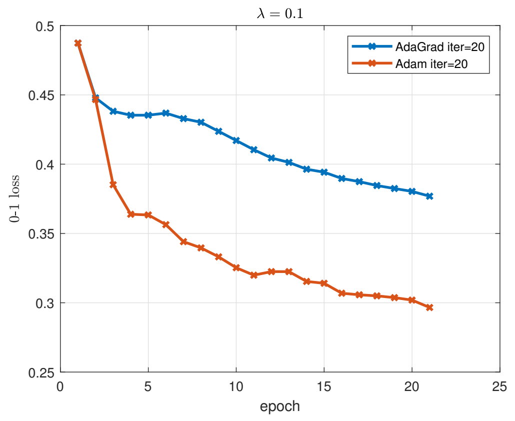
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

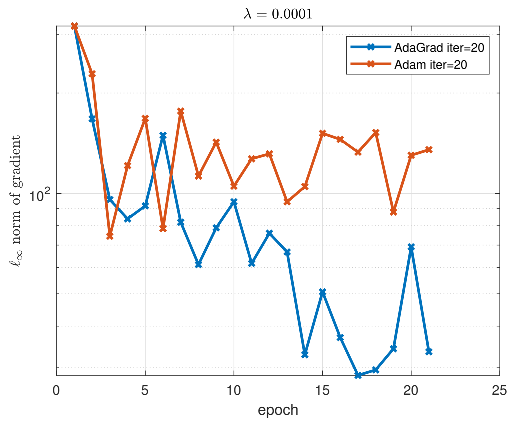


(c) $\lambda = 0.01$

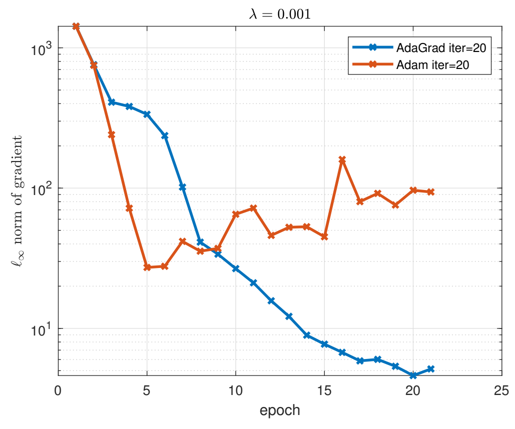


(d) $\lambda = 0.1$

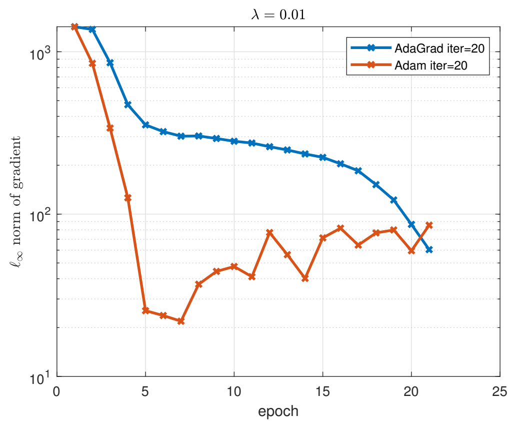
Figure III.21. 0-1 Loss on the MNIST model, batch size = 5000



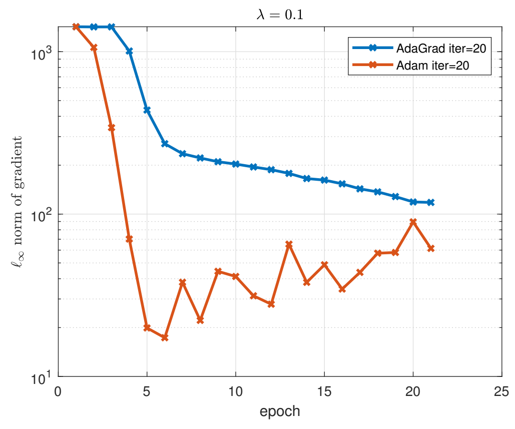
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

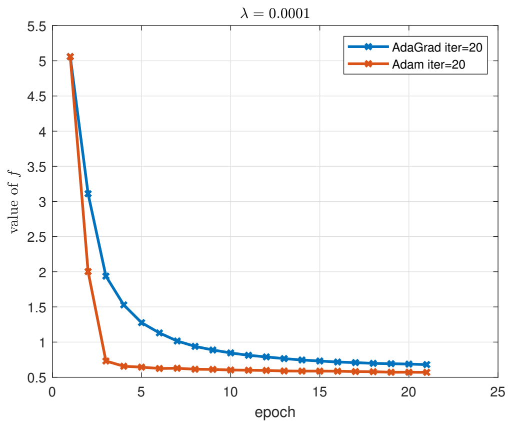


(c) $\lambda = 0.01$

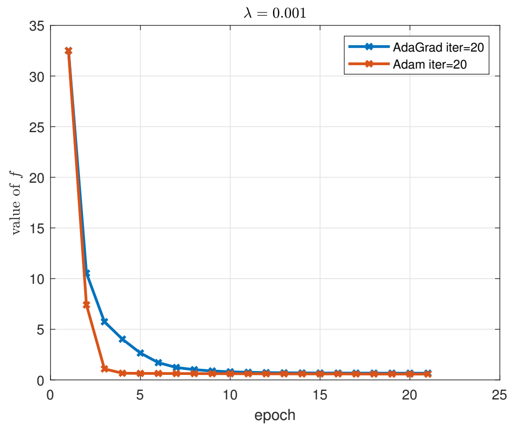


(d) $\lambda = 0.1$

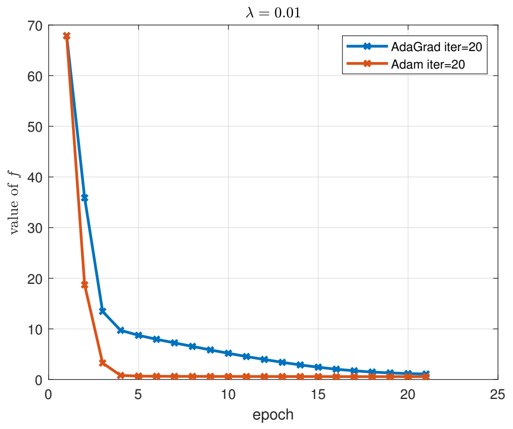
Figure III.22. Infinity norm of gradient on the MNIST model, batch size = 10000



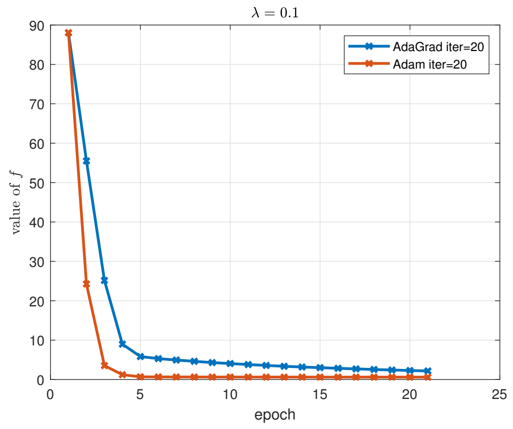
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$

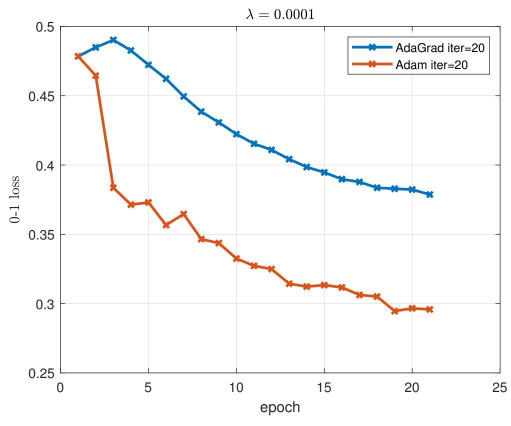


(c) $\lambda = 0.01$

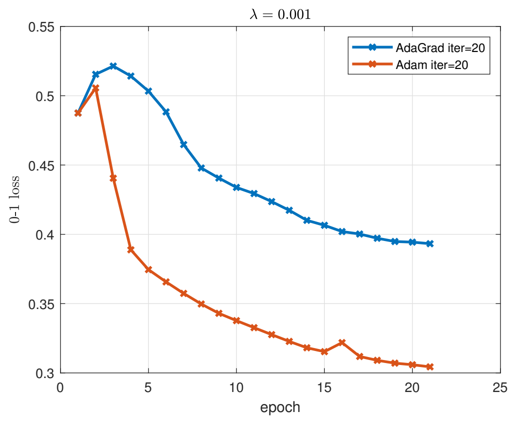


(d) $\lambda = 0.1$

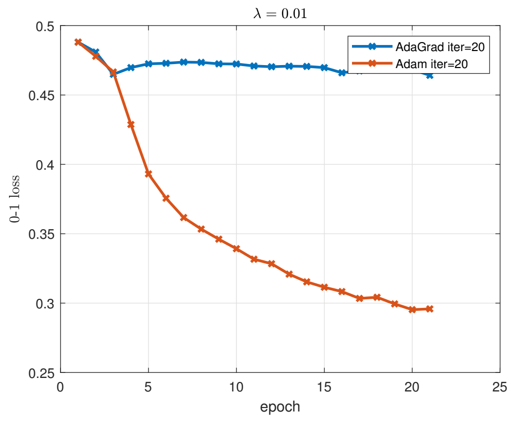
Figure III.23. Surrogate Loss on the MNIST model, batch size = 10000



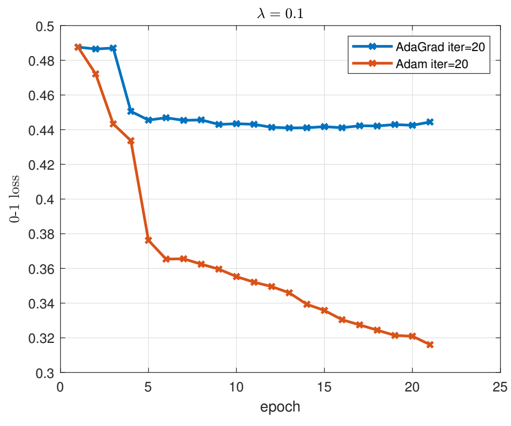
(a) $\lambda = 0.0001$



(b) $\lambda = 0.001$



(c) $\lambda = 0.01$



(d) $\lambda = 0.1$

Figure III.24. 0-1 Loss on the MNIST model, batch size = 10000

References

[Barzilai and Borwein, 1988] Barzilai, J. and Borwein, J. M. (1988). Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148.

[Blackard, 2000] Blackard, J. A. (2000). Comparison of neural networks and discriminant analysis in predicting forest cover types.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Tan et al., 2016] Tan, C., Ma, S., Dai, Y.-H., and Qian, Y. (2016). Barzilai-borwein step size for stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 685–693.