Variants of Stochastic Gradients Algorithms

AAIS in PKU   Weijie Chen 1901111420

April 22, 2020

1 Problem Settings

Considering the problem

$$\min_{w\in\mathbb{R}^d} \mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} f_i(w) + \lambda\|w\|_1 \tag{1}$$

where $f_i(w) = \log\left(1+\exp\left(-y^i w^\top x^i\right)\right)$, $(x^i, y^i) \in$Dataset (X,Y) and $\lambda > 0$.

In total, we use 2 ways to solve this problem, and we will show the algorithms, numerical results and their interpretation below. The 2 methods are:

- Adaptive gradient Method (Adagrad)
- Adaptive Moment Estimation Method (Adam)

1.1 Dataset

Dataset	Data Point n	Variable d	Reference
MNIST	70000	784	[1]
Covertype	581012	54	[2]

Table 1: Summary of dataset

We preprocess the above datasets to binary version with the following methods:

- MNIST: even for -1, odd for 1
- Covertype: type 2 for 1, otherwise for -1 [3]

2 Stochastic Gradients Algorithms

Adagrad is an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters.

Adam is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients r like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients s , similar to momentum.

Algorithm 1: Adaptive Gradient Method

```
Input  : Dataset $(X,Y)$ , learning rate $\alpha$, stop criteria $\epsilon$, hyperparameter $\delta$, batch size $N$
Output: iterations, optimal value
1  Initialize $w$ and $r$;
2  repeat
3      sample $N$ datapoint from $(X, Y)$;
4      $g \in \partial\mathcal{L}$ with batch samples;
5      $r := r + g \odot g$;
6      $w := w - \left(\frac{\alpha}{\delta+\sqrt{r}} \odot g\right)$;
7  until $\|\mathcal{L}_k - \mathcal{L}_{k+1}\| < \epsilon$;
```

Algorithm 2: Adaptive Moment Estimation Method

```
Input  : Dataset $(X,Y)$ , learning rate $\alpha$, stop criteria $\epsilon$, hyperparameter $\delta$, $\beta_1$, $\beta_2$, batch
         size $N$
Output: iterations, optimal value
1  Initialize $w$ ,$s$ and $r$;
2  repeat
3      sample $N$ datapoint from $(X, Y)$;
4      $g \in \partial\mathcal{L}$ with batch samples;
5      $s := \beta_1 s + (1 - \beta_1)g$;
6      $r := \beta_2 r + (1 - \beta_2)g \odot g$;
7      $\hat{s} = s/(1 - \beta_1^k)$;
8      $\hat{r} = r/(1 - \beta_2^k)$;
9      $w := w - \left(\frac{\alpha\hat{s}}{\delta+\sqrt{\hat{r}}}\right)$;
10 until $\|\mathcal{L}_k - \mathcal{L}_{k+1}\| < \epsilon$;
```

3 Numerical Result

We compare the performance of different optimization algorithms using iterations when achieving the same error. We use $\|\mathcal{L}_k - \mathcal{L}_{k+}\| < \epsilon$ as the stopping criteria. Therefore we can compare the convergence iterations of different methods in different λ settings. In practice, we set ϵ as 1e−4 for MNIST and Covertype dataset.

We test different $\lambda = 10, 1, 0.1, 0.001$ on different data set. Besides, we set the same random seed for both algorithms and datasets while training. The detail of the results can be seen in tables and figures.

From the numerical result, we have 2 obeservations as following:

- With different λ, performances of algorithms change a lot. λ is an indicator of the sparsity of w we want. When increase , the l_1-penalty will have more weight than the log-object function, it will weaken the smoothness of the total loss function and make w sparser.
- In general, Adam has better performance than Adagrad. But Adagrad is more stable than Adam while training.

Algorithms	Iterations	Optimal Loss
$\lambda = 10$		
Adagrad	6706	0.64155
Adam	129	0.69778
$\lambda = 1$		
Adagrad	97	0.52271
Adam	74	0.43653
$\lambda = 0.1$		
Adagrad	269	0.36076
Adam	88	0.32026
$\lambda = 0.001$		
Adagrad	300	0.33339
Adam	102	0.28509

Table 2: Numerical results for MNIST

Algorithms	Iterations	Optimal Loss
$\lambda = 10$		
Adagrad	7345	0.68777
Adam	90	0.78818
$\lambda = 1$		
Adagrad	231	0.68072
Adam	152	0.68108
$\lambda = 0.1$		
Adagrad	120	0.68696
Adam	94	0.67864
$\lambda = 0.001$		
Adagrad	111	0.69089
Adam	117	0.66206

Table 3: Numerical results for Covertype

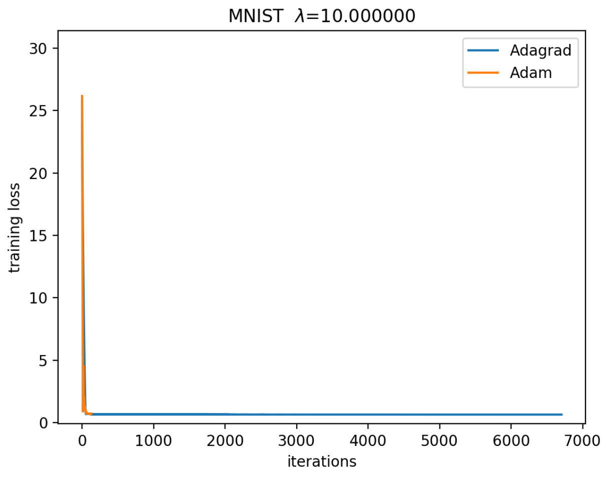

Figure 1: MNIST, $\lambda = 10$

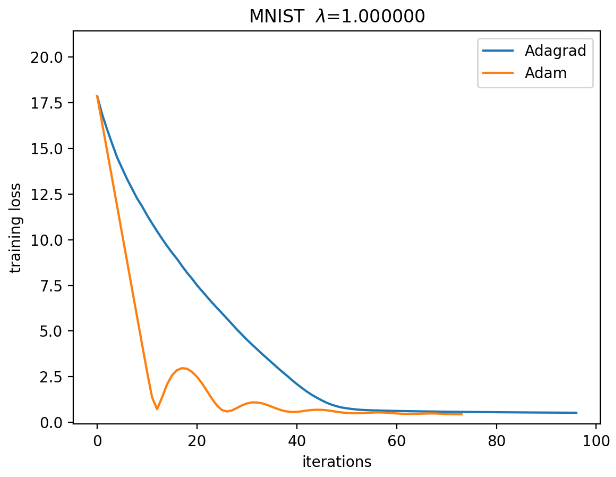

Figure 2: MNIST, $\lambda = 1$

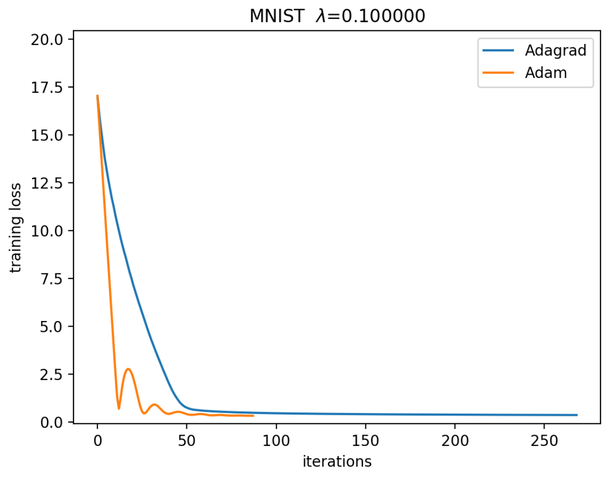

Figure 3: MNIST, $\lambda = 0.1$

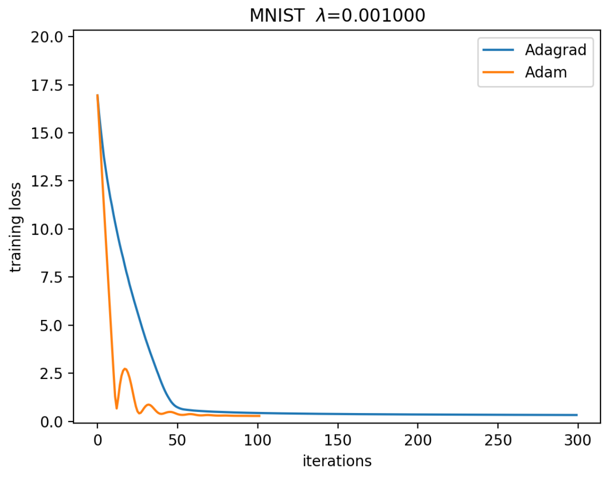

Figure 4: MNIST, $\lambda = 0.001$

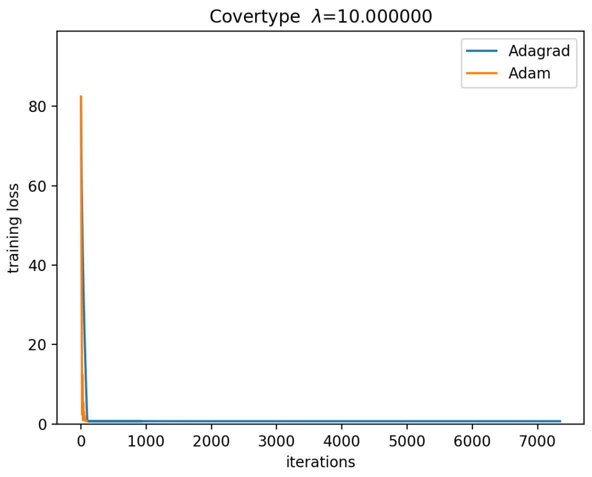

Figure 5: Covertype, $\lambda = 10$

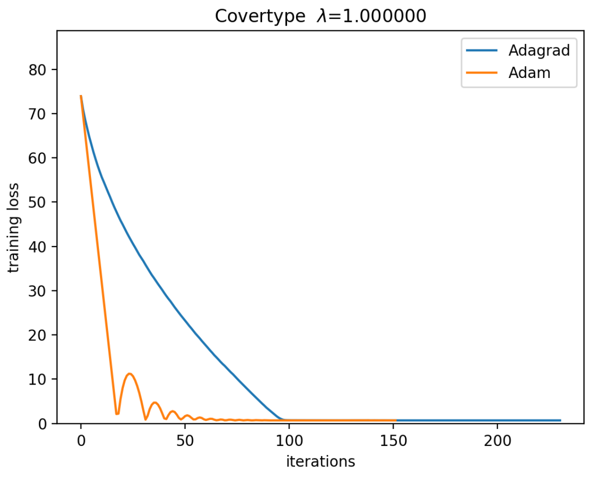

Figure 6: Covertype, $\lambda = 1$

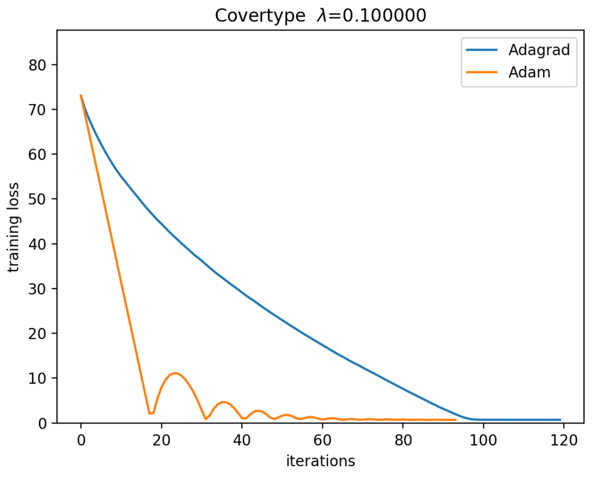

Figure 7: Covertype, $\lambda = 0.1$

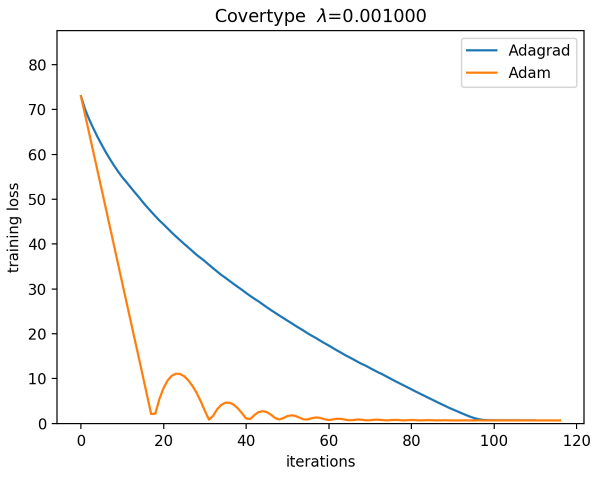

Figure 8: Covertype, $\lambda = 0.001$

4 README

The environment of numerical experiment as following:

- MacBook Pro(15-inch, 2019)
- 2.6 GHz 6-Core Intel Core i7
- 16 GB 2400 MHz DDR4
- macOS Catalina version 10.15.1
- python 3.6.9

Reference

[1] Yann LeCun, Corinna Cortes, and Christopher JC Burges. MNIST handwritten digit database. AT&T Labs [Online].
[2] Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. Computers and electronics in agriculture, 24(3):131–151, 1999.
[3] Collobert, Ronan, Samy Bengio, and Yoshua Bengio. ”A parallel mixture of SVMs for very large scale problems.” In Advances in Neural Information Processing Systems, pp. 633-640. 2002.