Stochastic Gradient Descent

Jeffrey Lu

May 1, 2025

Exercise 1

Given:

$$f(\theta) = \frac{1}{k}\sum_{j=1}^{k} Q(\theta, j) \quad \text{(average loss over all } k \text{ data points)}$$
$$g(\theta) = Q(\theta, i) \quad \text{where } i \sim \text{Uniform}(\{1, 2, \ldots, k\})$$
$$z = Q(\theta, i) - f(\theta) \quad \text{(noise)}$$

$\mathbb{E}[g(\theta)] = f(\theta)$

- The expected value of $g(\theta)$ is:
$$\mathbb{E}[g(\theta)] = \mathbb{E}[Q(\theta, i)]$$
- Since $i \sim \text{Uniform}(\{1, \ldots, k\})$, the probability of picking any index j is $\mathbb{P}(i = j) = \frac{1}{k}$.
- Therefore:
$$\mathbb{E}[Q(\theta, i)] = \sum_{j=1}^{k} \mathbb{P}(i = j) \cdot Q(\theta, j) = \sum_{j=1}^{k} \frac{1}{k} Q(\theta, j)$$
$$= \frac{1}{k}\sum_{j=1}^{k} Q(\theta, j) = f(\theta)$$

$\mathbb{E}[\nabla g(\theta)] = \nabla f(\theta)$

- $\nabla g(\theta) = \nabla Q(\theta, i)$
- $\mathbb{E}[\nabla g(\theta)] = \mathbb{E}[\nabla Q(\theta, i)]$
- Again, using the uniform distribution over $\{1, \ldots, k\}$:
$$\mathbb{E}[\nabla Q(\theta, i)] = \sum_{j=1}^{k} \mathbb{P}(i = j) \cdot \nabla Q(\theta, j) = \sum_{j=1}^{k} \frac{1}{k} \nabla Q(\theta, j)$$
$$= \frac{1}{k}\sum_{j=1}^{k} \nabla Q(\theta, j) = \nabla f(\theta)$$

$\mathbb{E}[z] = 0$

- By definition of noise:
$$z = Q(\theta, i) - f(\theta)$$
- Taking expectation (noting that $f(\theta)$ is not a random variable, but a fixed value):
$$\mathbb{E}[z] = \mathbb{E}[Q(\theta, i) - f(\theta)] = \mathbb{E}[Q(\theta, i)] - f(\theta)$$
- From earlier, $\mathbb{E}[Q(\theta, i)] = f(\theta)$, so:
$$\mathbb{E}[z] = f(\theta) - f(\theta) = 0$$

Exercise 2

	SGD	**GD**
Computational Cost per Update	$O(1)$	$O(k)$
Number of Updates to Reach ϵ	$O\left(\frac{1}{\epsilon}\right)$ (Theorem 1.1)	$O\left(\ln\left(\frac{1}{\epsilon}\right)\right)$ (Theorem 1.3)
Total Cost	$O\left(\frac{1}{\epsilon}\right)$	$O\left(k \cdot \ln\left(\frac{1}{\epsilon}\right)\right)$

Explanation

- **Cost per update:**
  - SGD uses one random data point per update: $O(1)$. Specifically, as given, $\nabla g(\theta)$ takes $O(c)$ or $O(1)$ time.
  - GD uses all k data points per update: $O(k)$. Specifically, as given, $\nabla f(\theta)$ takes $O(kc)$ or $O(k)$ time.
- **Number of updates to reach error ϵ:**
  - SGD under ideal assumptions (Theorem 1.1) as f is assumed strongly convex and twice continuously differentiable: $O(1/\epsilon)$
  - GD (Theorem 1.3): $O(\ln(1/\epsilon))$
- **Total cost:**
  - SGD: $O(1/\epsilon) \cdot O(1) = O(1/\epsilon)$
  - GD: $O(\ln(1/\epsilon)) \cdot O(k) = O(k \ln(1/\epsilon))$

Exercise 3

	SGD	**GD**
Computational Cost per Update	$O(1)$	$O\left(\rho^{-1/\gamma}\right)$
Number of Updates to Reach ρ	$O\left(\frac{1}{\rho}\right)$	$O\left(\ln\left(\frac{1}{\rho}\right)\right)$
Total Cost	$O\left(\frac{1}{\rho^{1+\gamma}}\right)$	$O\left(\rho^{-1/\gamma}\ln\left(\frac{1}{\rho}\right)\right)$

Explanation

- **Cost per update:**
  - Same as before: SGD is $O(1)$, GD is $O(k)$.
- **Number of updates:**
  - From Exercise 2, number of updates depends on training loss ϵ.
  - We assume $\epsilon = \Theta(\rho)$, so number of updates behaves similarly.
  - Thus:
  $$\text{SGD: } O\left(\frac{1}{\rho}\right), \quad \text{GD: } O\left(\ln\left(\frac{1}{\rho}\right)\right)$$
- **Total cost:**
  - To achieve a precision ρ, you need $k = O(\rho^{-1/\gamma})$ training samples, because $\rho = O(k^{-\gamma})$.
  - So:
    * For SGD:
    $$\begin{aligned}\text{Total cost} &= \text{updates} \times \text{cost per update}\\ &= O\left(\frac{1}{\rho}\right) \times O(1) = O\left(\frac{1}{\rho}\right)\end{aligned}$$
    But since k must also be $O(\rho^{-1/\gamma})$, and each sample is independent, the overall cost becomes:
    $$O\left(\frac{1}{\rho^{1+\gamma}}\right)$$
    * For GD:
    $$\text{Total cost} = O\left(\ln\left(\frac{1}{\rho}\right) \times k\right)$$
    and since $k = O(\rho^{-1/\gamma})$:
    $$= O\left(\ln\left(\frac{1}{\rho}\right) \times \rho^{-1/\gamma}\right)$$

Why GD can be bad

When γ is small (like 0.2):

- $k = O(p^{-5})$
- $\rho^{-1/\gamma}$ grows very fast.
- GD must process the full training set at every step - extremely expensive. For example, if $p = 0.01$, then you would need $k = 0.01^{-5} = 10000000000$ samples, which is very expensive to compute for GD especially.
- SGD can handle the larger training set much more efficiently by using a much smaller, albeit noisy sample.
- Although for higher γ the two total cost equations begin to equalize, for lower γ SGD outperforms GD loss-wise

Thus, choosing GD when γ is small becomes very inefficient.

Exercise 4

Question 1

When $\gamma = 2$, the loss function becomes:

$$Q(\theta, j) = \left(x_j^T \theta - y_j\right)^2$$

Thus, the total loss over the training set is the Mean Squared Error (MSE):

$$F(\theta) = \frac{1}{k} \sum_{j=1}^{k} \left(x_j^T \theta - y_j\right)^2$$

The closed-form solution that minimizes the MSE is the standard least squares estimator:

$$\theta = (X^T X)^{-1} X^T y$$

where:

- X is the $k \times d$ matrix of input vectors x_j,
- y is the $k \times 1$ vector of labels y_j,
- $d = 10$ is the dimensionality of x_j.

Implementing, we get:

```
Empirical theta = [1.00941229 1.00225733 1.00662721 1.00484361 1.00523307 1.00247492
 0.99779679 1.01149235 1.000694    1.00713222]
```

This is very close to the true values of $\theta_{true} = 1$

Question 2

Given the general form:

$$Q(\theta, j) = \left|x_j^T \theta - y_j\right|^\gamma$$

Define:

$$e_j(\theta) = x_j^T \theta - y_j \quad \text{(the prediction error)}$$

Applying the chain rule, the gradient of $Q(\theta, j)$ with respect to θ is:

$$\nabla Q(\theta, j) = \gamma \, |e_j(\theta)|^{\gamma - 1} \, \text{sgn}(e_j(\theta)) \, x_j$$

Thus, the final expression is:

$$\nabla Q(\theta, j) = \gamma \, \left|x_j^T \theta - y_j\right|^{\gamma - 1} \, \text{sgn}\left(x_j^T \theta - y_j\right) \, x_j$$

Question 3

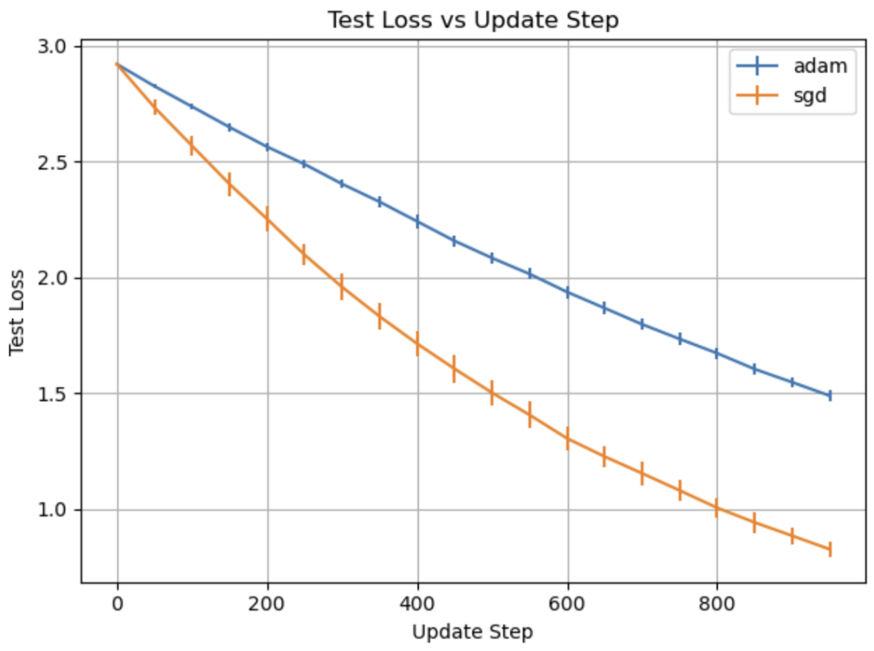
Final parameters:

```
ADAM: [0.30072142 0.33621711 0.35234173 0.28647327 0.32207288 0.37785626
 0.35936013 0.35300483 0.38980936 0.3715772 ]

SGD: [0.69075776 0.77598346 0.74513576 0.70377153 0.80302679 0.74187181
 0.65412221 0.70737926 0.76404636 0.79229806]
```

It appears that both algorithms are quite far from the true $\theta_{true} = 1$, although SGD seems to have converged faster. This is most probably because there is not enough updates, as 1000 is not that many updates. If we increased this to a much larger value like 10000, we would see θ values much closer to the true value of 1.

Question 4



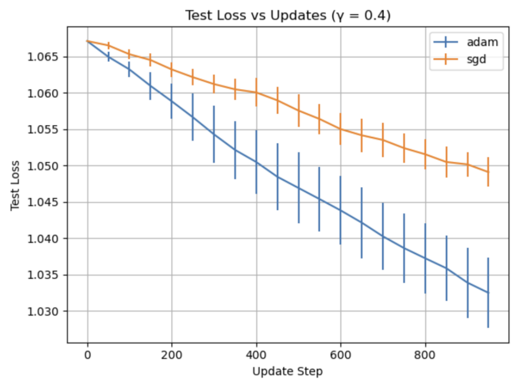
It seems that SGD outperforms ADAM in terms of loss at a $\gamma = 2$. The error bars show statistically significant difference in loss between the two algorithms.

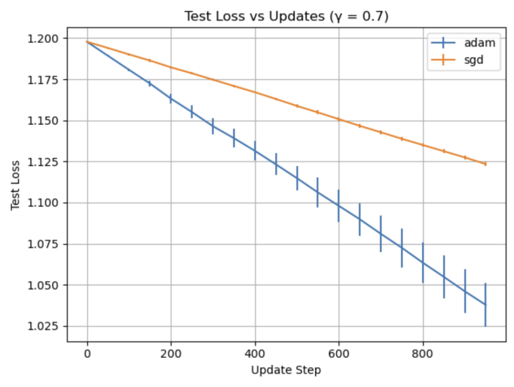
Question 5

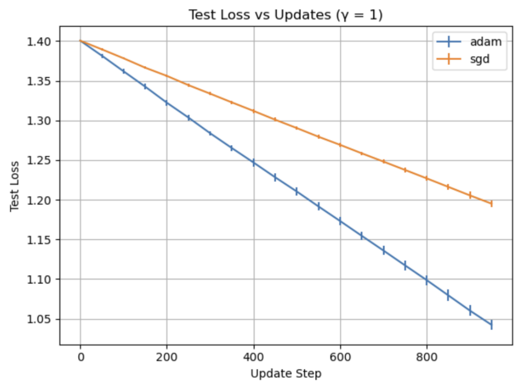
ADAM consistently outperforms SGD at lower learning rates, when $\gamma = 0.4$, 0.7, and 1. In these settings, ADAM achieves significantly lower loss values and demonstrates faster, more stable convergence across training iterations. This performance stems from ADAM's core design: it combines the benefits of both momentum and adaptive learning rates by maintaining exponentially decaying averages of past gradients (first moment) and squared gradients (second moment). These estimates help ADAM adjust the learning rate dynamically for each parameter, allowing it to take larger steps when gradients are small and smaller, more cautious steps when gradients are large. This adaptive behavior is especially advantageous in scenarios where the loss surface is noisy or has varying (non-convex) curvature, as ADAM can navigate such landscapes more efficiently without manual tuning of the learning rate for different problem regions.

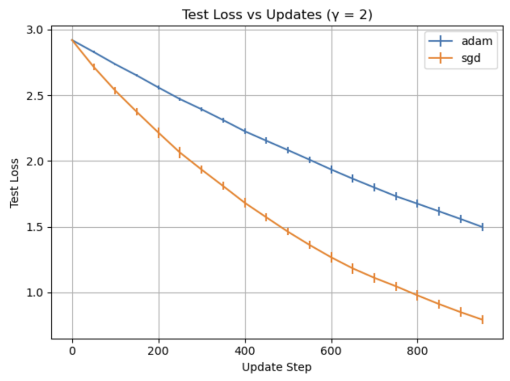
SGD, by contrast, uses a fixed learning rate for all parameters and all iterations. At lower learning rates, this leads to slow convergence, as the step size is uniformly small regardless of how flat or steep the gradient is. Without adaptive scaling, SGD may require significantly more iterations to reach a comparable solution-or may get stuck in regions of low gradient magnitude (plateaus). It lacks the momentum and adaptivity to adjust its trajectory effectively in more complex or ill-conditioned optimization landscapes. However, for higher learning rates such as $\gamma = 2$, 3, and 5, ADAM's adaptivity can harm it. Because it compounds gradient updates through its moment estimates, a large base learning rate can amplify updates too aggressively, leading to instability or divergence. In these cases, SGD's simpler, more generalized update rule becomes more robust. Since it applies gradients directly with a constant scale, it is less sensitive to compounded variance in updates and may maintain more stable behavior, even if convergence is slower.

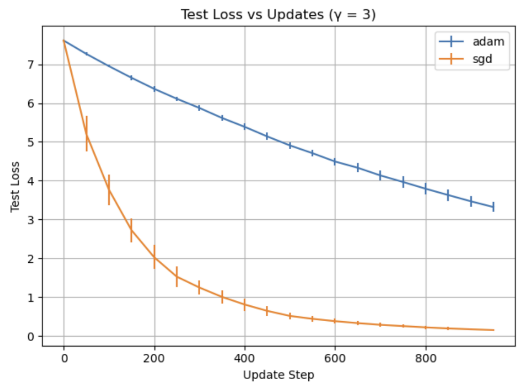
The graphs for $\gamma = 0.4, 0.7, 1, 2, 3, 5$ are attached as follows, and we can see the above analysis present. None of the error bars overlap, showing statistical significance in the results.

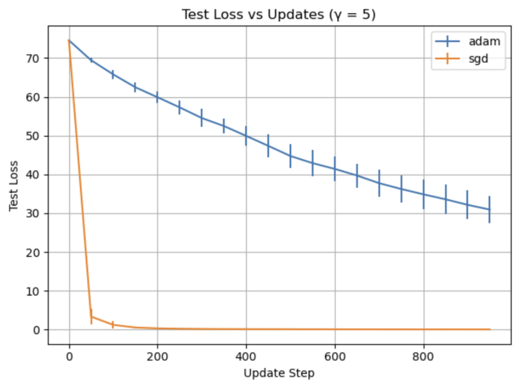

Test Loss vs Updates (γ = 0.4)
adam
sgd
Test Loss
Update Step
1.065
1.060
1.055
1.050
1.045
1.040
1.035
1.030
0
200
400
600
800


Test Loss vs Updates (γ = 0.7)
adam
sgd
Test Loss
Update Step
1.200
1.175
1.150
1.125
1.100
1.075
1.050
1.025
0
200
400
600
800


Test Loss vs Updates (γ = 1)
adam
sgd
Test Loss
Update Step
1.40
1.35
1.30
1.25
1.20
1.15
1.10
1.05
0
200
400
600
800


Test Loss vs Updates (γ = 2)
adam
sgd
Test Loss
Update Step
3.0
2.5
2.0
1.5
1.0
0
200
400
600
800


Test Loss vs Updates (γ = 3)
adam
sgd
Test Loss
Update Step
7
6
5
4
3
2
1
0
0
200
400
600
800


Test Loss vs Updates (γ = 5)
adam
sgd
Test Loss
Update Step
70
60
50
40
30
20
10
0
0
200
400
600
800

Exercise 5

Question 1

- Gradient Descent computes the gradient using the entire training dataset. When we set the SGD batch size to 60,000 (the full dataset), the optimizer computes the exact same full gradient at every step, which is exactly what GD does.
- The accuracy is 0.1408. This is a fairly low accuracy. The loss over the epochs also did not change much, starting at a loss of 2.3036 after epoch 0 to just 2.2954 after epoch 14. This suggests that the model is struggling to learn - likely due to poor gradient updates. GD can suffer from very stable but small updates that make convergence extremely slow, especially on high-dimensional, non-convex problems like neural networks. Additionally, GD does not update that often (only one time per epoch), resulting in less chances to descend the loss function towards a minima.
- Two ways to improve the accuracy are:
  - Use a smaller batch size : This converts the optimizer into SGD, which introduces randomness in updates. This noise helps the model explore the loss surface better and escape sharp or flat regions, improving both convergence speed and test accuracy.
  - Tune the learning rate: The learning rate controls how large each update step is. A rate that is too small can lead to very slow learning and early stagnation, as seen in our model. Carefully tuning the learning rate or using a learning rate scheduler (via models like ADAM) can significantly improve convergence and final accuracy.

Question 2

- The accuracy is 0.9267. This result shows that the model trained using SGD was able to achieve strong generalization on the MNIST dataset. Compared to full-batch Gradient Descent, SGD converged faster and achieved significantly higher accuracy, likely due to the stochasticity introduced by smaller batch updates, which helps the model escape sharp minima and overfit less, while updating more often.
- One way to further improve accuracy is to use momentum and adaptive learning rates. The momentum term allows an algorithm to remember past gradients, helping it smooth out noisy updates and accelerate learning in the right direction. Adaptive learning rate mechanisms adjust step sizes individually for each parameter, allowing faster convergence on sparse or complex loss surfaces and more fine-tuned convergence of the parameters in more plateaued surfaces.

Question 3

- The accuracy is 0.9765. This is higher than the accuracy achieved using full-batch Gradient Descent or SGD. The model converged quickly within a few epochs (around 3), and the training loss dropped sharply compared to previous methods.
- ADAM converges faster than SGD because it combines momentum and adaptive learning rates as commented on in the previous question. Momentum helps accelerate updates in consistent gradient directions and dampens oscillations. Meanwhile, adaptive learning rates scale each parameter update in relation to the recent gradient variance, which ensures more stable updates. This allows ADAM to take large, confident steps early and smaller, precise ones near convergence - leading to faster training and better generalization. Essentially, it converges fast at the start, then takes cautious steps to fine-tune and get higher accuracy near the end.