Why Considering Second Order Gradient is Useless

Kaizhao Liang

December 2025

1 Introduction

Consider vanilla SGD with bi-level objective, basically GD on x should also

$$\theta^{'} = \theta - \lambda \frac{\partial \mathcal{L}(\theta, x)}{\partial \theta} \tag{1}$$

$$\frac{\partial \mathcal{L}(\theta^{'}, x')}{\partial \theta} = (1 - \lambda \frac{\partial^2 \mathcal{L}(\theta, x)}{\partial^2 \theta}) \cdot \frac{\partial \mathcal{L}(\theta', x')}{\partial \theta'} \tag{2}$$

As λ is small and second order gradients are in general bounded (smoothness assumption)

$$\frac{\partial \mathcal{L}(\theta^{'}, x')}{\partial \theta} \approx \frac{\partial \mathcal{L}(\theta, x')}{\theta} \tag{3}$$

The second order objective is equivalent to running first order gradient on x'. Thus, **a simple pretraining objective suffices to induce the online learning ability in the ideal world.**

This becomes a bit more interesting (complex) when other optimizers are considered, where the update rule becomes something like:

$$\theta^{'} = \theta - \lambda \phi(\frac{\partial \mathcal{L}(\theta, x)}{\partial \theta}) \tag{4}$$

$$\frac{\partial \mathcal{L}(\theta^{'}, x')}{\partial \theta} = (1 - \lambda \frac{\partial}{\partial \theta} \phi(\frac{\partial \mathcal{L}(\theta, x)}{\partial \theta})) \cdot \frac{\partial \mathcal{L}(\theta', x')}{\partial \theta'} \tag{5}$$

This is particularly advantageous for optimizers like Muon, where the derivative of $\phi = msign(\cdot)$ is explicitly bounded (roughly by $\frac{1}{m}$, where m is the matrix size) regardless of the landscape of $\mathcal{L}$. That also sort of explains why the model trained with Muon seems to exhibit better generalization capability.

The above insight is not really actionable, but we could derive a useful criterion for success for any pretraining run. **A well-trained model should automatically be a lifelong learner via gradient descent**. A simple test to evaluate any of the pretrained models can be finetuning it to remember some arbitrary facts (such as passwords), meanwhile it does not degrade on other tests.

I am speculating the current models only scratch the surface of pretraining and have not reached the point where the loss landscape is flat everywhere. **We should just keep training, even training loss doesn't go down (significantly).**

2 The Min-Min Algorithm

Although scaling pretraining will eventually emerge life-long learning behavior naturally, we might want to design algorithms to speed up this process due to limitations such as the amount of data and compute available to us. This is very much akin to the Lookahead Optimizer, except that in the outer loop there is an extra step of backpropagation.

Algorithm 1: Min–Min (Two-Stage Minimization with Small/Large Batches)

```
Input  : parameters $\theta_0$; learning rate $\eta$; inner step size $\rho$; small-batch
         size $b_s$; large-batch size $b_\ell$ ($b_s \leq b_\ell$); loss $\ell(\theta; z)$; number of
         steps $T$
Output: final parameters $\theta_T$
for $t = 0, 1, \ldots, T-1$ do
    Sample a small batch $B_s^{(t)}$ of size $b_s$
    Compute small-batch gradient $g_s \leftarrow \nabla_\theta \left( \frac{1}{|B_s^{(t)}|} \sum_{z \in B_s^{(t)}} \ell(\theta_t; z) \right)$
    // Inner minimization step (first minimization, small
       batch)
    if $g_s \neq 0$ then            // optional, SAM-like normalization
        $\delta_t \leftarrow -\rho \frac{g_s}{\|g_s\|_2}$
    else
        $\delta_t \leftarrow 0$
    $\tilde{\theta}_t \leftarrow \theta_t + \delta_t$
    Sample a large batch $B_\ell^{(t)}$ of size $b_\ell$, typically $B_\ell^{(t)} \cap B_s^{(t)} = \emptyset$
    // Outer minimization step (second minimization,
       larger/different batch)
    $g_\ell \leftarrow \nabla_\theta \left( \frac{1}{|B_\ell^{(t)}|} \sum_{z \in B_\ell^{(t)}} \ell(\tilde{\theta}_t; z) \right)$
    $\theta_{t+1} \leftarrow \theta_t - \eta \, g_\ell$
return $\theta_T$
```