Thesis Notepad of Ideas and Progress etc

June 26, 2025

Contents

1 Abstract **5**

2 Introduction **7**

3 Understanding Adam **9**
3.1 Preliminaries and Related Work . . . 9
3.2 Problem Settings and Experimental Details . . . 10
3.3 Some Experiments/Ablations (Mostly an experiment graveyard sadly) . . . 10
3.3.1 Nesting the Moving averages . . . 10
3.3.2 Adam2SGD and ε schedule . . . 11
3.3.3 How EMA smooths the update signal . . . 11
3.3.4 BackPACK . . . 11
3.3.5 The ε Hyperparameter . . . 11

4 Bias Correction in Adam: A Detailed Analysis **13**
4.1 Bias Correction: An Unnecessary Evil? . . . 13
4.1.1 A Discussion of the "Proof" for Bias Correction . . . 14
4.2 Experimental Outline . . . 14
4.3 Is Bias Correction just Hidden Learning rate scheduling? . . . 17

5 Beyond Adam: Is Muon the future? **21**
5.1 Related Work . . . 22
5.2 Understanding Muon . . . 22
5.2.1 Metrized Deep Learning . . . 22
5.2.2 Dualized Gradients and Metrized Deep Learning . . . 22
5.2.3 Proof of Orthogonal Update step . . . 22
5.2.4 Muon's Relationship to Shampoo . . . 22
5.2.5 The Newton Schulz Algorithm . . . 22
5.3 Ablations on Simple Problems . . . 22
5.3.1 Linear Regression . . . 23
5.3.2 A Quadratic Problem . . . 24
5.3.3 Logistic Regression . . . 24
5.3.4 Small MLPs . . . 25
5.4 SVD structural analysis . . . 25
5.4.1 CIFAR-10 ResNet . . . 25

5.5 Experimental 26

6 Discussion and Future Work **27**

A How Exponential Moving Averages Smooth the Update Signal **31**

Chapter 1

Abstract

The performance of deep learnings model is strongly influenced by the choice optimization algorithm used during training. While Adam with decoupled weight decay has has become the de facto standard for training large-scale models, certain components of the optimizer remain only partially understood.

In the first part of this thesis, we conduct a series of empirical investigations with the aimed at identifying which elements of Adam are critical to its strong performance. In particular, we perform ablations on the ordering of the exponential moving averages, the role of the ε hyperparameter and offer a novel interpretation of bias correction. Notably we demonstate that the inclusion of bias correction in the commonly used setting of $\beta_1 = \beta_2 = 0.95$ in language models, the inclusion of bias correction can actually impair performance.

In the second part of the thesis, we explore the recently proposed Muon optimizer. In an attempt to understand the minimal conditions for which Muon offers a practical advantage over SGD and Adam, we study a number of simple problem settings such as multi-output linear regression and a generalized quadratic problem.

Additionally, we perform ablations on a Resnet implementation on CIFAR-10, studying the Newton Schulz polynomial which approximates the gradient orthogonalization and investigate whether applying the approximation to an exponential moving average is truly preferable to individual components.

Chapter 2

Introduction

TODO: Ask about whole story telling element and how i should frame the headless chicken with failing ideas approach i regretably followed
Optimization algorithms are

Chapter 3

Understanding Adam

The Adam optimizer [KB17] was first proposed just over a decade ago. With the integral addition of decoupled weight decay [LH19], it has become the de facto stochastic optimizer choice for nearly all SOTA deep learning training, particularly in the pretraining of language models. Extensive research into understanding Adam's dynamics has been conducted however several aspects are not completely understood.

The optimizer itself is simply a synthesis of two already well-established optimization ideas - momentum and RMSProp. Adam inherits the stabilizing effect of momentum and the adaptive scaling of RMSProp. Despite extensive research into optimizers for deep learning in the past decade, AdamW remains the default choice for the vast majority of both industry and academic applications.
In this chapter, we discuss

3.1 Preliminaries and Related Work

For signal $(s_j)_{j\geq 1}$ and $0 \leq \beta < 1$, we define the β-exponential moving average $\mathrm{EMA}_\beta(s_t)$ by

$$\mathrm{EMA}_\beta(s_t) := \begin{cases} 0 & \text{if } t = 0 \\ \beta\, \mathrm{EMA}_{t-1}(s_{t-1}) + (1-\beta)s_t, & \text{if } t \geq 1 \end{cases}$$

Or as a closed form expression:

$$\mathrm{EMA}_\beta(s_t) := (1-\beta)\sum_{j=1}^{t} \beta^{t-j} s_j$$

In this notation, one can then express (without bias correction) Adam optimizer [1] as:

$$\theta_{t+1}' = \theta_t - \eta_t \frac{\mathrm{EMA}_{\beta_1}(g_t)}{\sqrt{\mathrm{EMA}_{\beta_2}(g_t^2)} + \varepsilon},$$

---

[1] The update for AdamW (Adam with decoupled weight decay) is given by $\theta_{t+1} = (1 - \eta_t \lambda)\theta$

where $\theta_t, g_t \in \mathbb{R}^d$ are the respective model parameters and gradients at iteration $t \geq_1$, $(\eta_j)_{j\geq 1}$ the scheduled learning rate and $\varepsilon \in \mathbb{R}_{>0}$ is a small constant **TODO: Talk about the papers about understanding Adam and then a bit about what is stil not understood/ bottlenecks/is it really the best we can do**

3.2 Problem Settings and Experimental Details

EAdam Paper probably

Relation to signSGD

Why transformers need Adam: hessian and class imbalance

TODO: Talk about this latest batch size SGD paper from group that challenges these prior explanations

The Secret Sauce paper

3.3 Some Experiments/Ablations (Mostly an experiment graveyard sadly)

'

- Nested moving average vs regular Adam: are they the same? Interesting since the choice of whichway to take the EMA is sort of arbitrary.
- Adam2SGD: Could look at SGD2Adam since the paper from Teodora showed some advantage in doing so
- Epsilon scheduling experiments: could be very related to the sgd thing... Can also be bespoke to parameter (seems like a universal epsilon for all parameters could be foolish.
- Beta scheduling too with experiments about the angle between the two
- The BACKPack package experiment with class variances individual. Relationship to signSGD stuff from Hennig paper.

3.3.1 Nesting the Moving averages

Coupling of momentum and RMSProp

Adam is given by:

$$w^{t+1} = w^t - \eta \frac{\mathrm{EMA}_{\beta_1}(g_t)}{\sqrt{\mathrm{EMA}_{\beta_2}(g_t^2)} + \varepsilon}$$

In other words, Adam combines the The update direction $\frac{\mathrm{EMA}_{\beta_1}(g_t}{\sqrt{\mathrm{EMA}_{\beta_2}(g_t^2)+\varepsilon}}$ one could also consider the double nesting formula

$$w^{t+1} = w^t - \eta \mathrm{EMA}_{\beta_1}\left(\frac{g_t}{\sqrt{EMA_{\beta_2}(g_t^2)+\varepsilon}}\right)$$

Or expressed in iterative steps:

$$v^t = \beta_2 v^{t-1} + (1-\beta_2) g_t^2$$
$$s^t = \beta_1 s^{t-1} + (1-\beta_1)\frac{g_t}{\sqrt{v^t}+\varepsilon}\theta_t = \theta$$

In 4 we demonstrate in detail that the significance of bias correction in AdamW overstated, particularly when appropriate learning rate scheduling is conducted. We therefore include no bias correction terms.

3.3.2 Adam2SGD and ε schedule

TODO: Ask Antonio if this is just fucked/should go in the appendix/nowhere

3.3.3 How EMA smooths the update signal

TODO: Add this angle between gradients and updates vs update and previous update experiment with plots and actually do the beta scheduling thing you keep thinking/talking about!

3.3.4 BackPACK

TODO: Talk about the shitty variance KL divergence experiment to see if v_t is somehow a proxy for variance. Also the whole optimal batch size scaling based on the variance of the mini-batch

3.3.5 The ε Hyperparameter

While regular adam is given by:

$$w^{t+1} = w^t - \eta\frac{\mathrm{EMA}_{\beta_1}(g_t)}{\sqrt{\mathrm{EMA}_{\beta_2}(g_t^2)}+\varepsilon}$$

with $\varepsilon = 1e-8$ as the default choice.

It has been observed in a number of papers ([YG20]), that language model pretraining can be improved by altering the epsilon hyperparameter. Indeed increasing ε to $1e-6$ can decrease final validation perplexity but at the cost of decreased training stability. Conversely, decreasing ε can improve stability without noticable decrease in performance. In every one of these experiments, the ε value is chosen to apply globally, however the We consider the distribution of

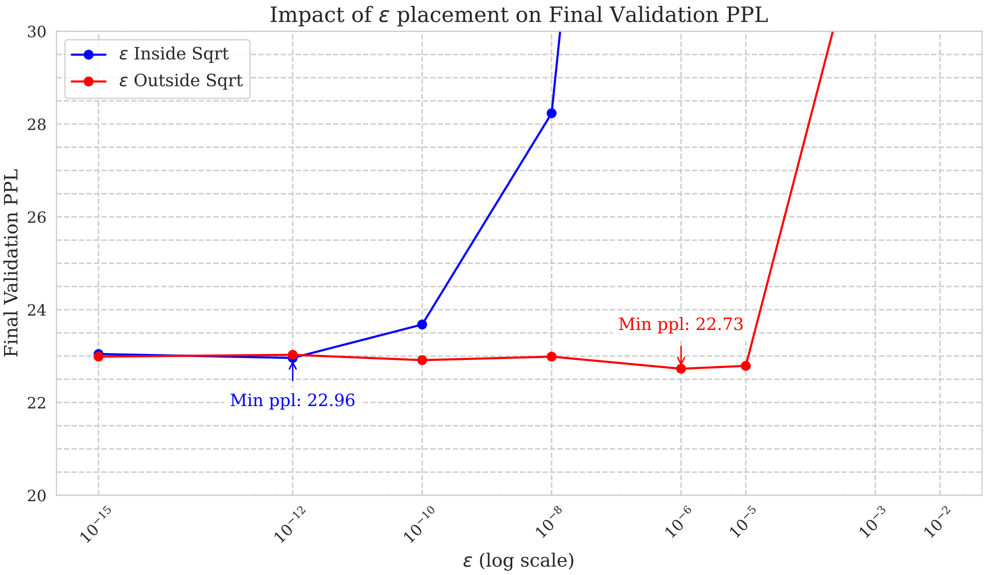


Figure 3.1: Sensitivity of AdamW's final validation performance (100M held-out tokens) to ϵ placement on 162M parameter transformer. Compares ϵ inside square root ($\frac{m_t}{\sqrt{v_t+\varepsilon}}$) versus outside ($\frac{m_t}{\sqrt{v_t}+\varepsilon}$). Averaged across 3 random seeds. The minima for each configuration are annotated. All other hyperpararameters are

We observe that placing ε inside square root is more sensitive to the choice of ε but optimal performance is still similar.

Chapter 4

Bias Correction in Adam: A Detailed Analysis

4.1 Bias Correction: An Unnecessary Evil?

The AdamW optimizer can be represented as follows:(**TODO: Is algorithm or plain equation nicer? Everyone obviously knows AdamW but maybe it's nice to make clear the bias correction being put in/out?**

Algorithm 1 AdamW Optimizer (Bias Correction Experimental Focus)

```
Require: $(\eta_t)_{t \geq 1} \subset \mathbb{R}_{\geq 0}, \beta_1, \beta_2 \in [0, 1), \lambda \in \mathbb{R}_{>0}, \epsilon \in \mathbb{R}_{>0}$    ▷ Standard hyperparameters
Require: $\theta_0 \in \mathbb{R}^d$    ▷ Initialised model weights
Require: do_bias_correction: bool    ▷ Experimental flag
Initialize $m_0, v_0, t \leftarrow 0$
while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
    if do_bias_correction then
        $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ , $v_t \leftarrow \frac{v_t}{1-\beta_2^t}$    ▷ bias correction not by default
    end if
    $\theta_t \leftarrow (1 - \eta_t \lambda) \theta_{t-1} - \eta_t \frac{m_t}{\sqrt{v_t} + \epsilon}$
end while
```

$$
\begin{aligned}
m^{t+1} &= \beta_1 m^t + (1 - \beta_1) g^t \quad &, \quad v^{t+1} &= \beta_2 v^t + (1 - \beta_2) g^t \odot g^t \\
\hat{m}^{t+1} &= \frac{1}{1 - \beta_1^{t+1}} m^{t+1} \quad &, \quad \hat{v}^{t+1} &= \frac{1}{1 - \beta_2^{t+1}} v^{t+1} \\
w^{t+1} &= (1 - \eta_t \alpha) w^t - \eta_t \frac{\hat{m}^{t+1}}{\sqrt{\hat{v}^{t+1}} + \varepsilon}
\end{aligned}
$$

Where $\eta_t \in \mathbb{R}_{\geq 0}$ is the learning rate at time step t, $\alpha \in \mathbb{R}_{\geq 0}$ is the weight decay and $\beta_1, \beta_2, \varepsilon$ are the model's hyperparemeters.
Often research papers relating to Adam(W) ([GBC16]), include the bias correction term in their analyses. Indeed bias correction is generally viewed by the optimization community as an inexorable component of the Adam(W) optimizer
The purpose of this chapter is to discuss the role of bias correction on the performance of models trained using AdamW. For the purposes of all experiments conducted, we make use of a custom implementation of the AdamW optimizer.

4.1.1 A Discussion of the "Proof" for Bias Correction

In a number of pedagogical resources ([GBC16], Deep Learning Lecture), blogs and indeed the original Adam paper ([KB17] - some variant of the following "proof" is given to justify the inclusion of bias correction:

$$\begin{aligned}\mathbb{E}[m_t] &= \mathbb{E}\left[(1-\beta_1)\sum_{j=1}^{t}\beta_1^{t-i}g_i\right] \\ &= \mathbb{E}\left[(1-\beta_1)g_t\sum_{j=1}^{t}\beta_1^{t-i}\right] \text{ (assuming } \mathbb{E}[g_t] \approx E[g_i]\forall i < t) \\ &= (1-\beta_1^t)\mathbb{E}[g_t]\end{aligned}$$

It is therefore argued that dividing by $1-\beta_1^t$ removes the expected "bias" in the exponential moving average.
Where an analogous argument is used to justify the bias correction factor for v_t.
This assumption that $\mathbb{E}[g_t] \approx \mathbb{E}[g_i]$ for $i < t$ is blatently false. We conduct a number of experiments with a custom implementation of AdamW and investigate the effect of this term.
TODO: some visualiation or way to make this look nice

4.2 Experimental Outline

We conduct a number of experiment in both a vision and language setting.
In the vision setting, we consider:

- A multi-layered perceptron (MLP) and simple convolutional neural network (CNN) on the MNIST, Fashion MNIST and SVHN.
- A less overparametrized ResNet implementation on the CIFAR10 dataset (only $\sim$ 1M parameters rather than ResNet 18 which has $\sim$ 11M)
- Both a ResNet56[1], Densenet121 [HLvdMW18] and a vision transformer on the CIFAR100 dataset

---

[1] Pytorch's ResNet50 was specifically designed for Imagenet's 224 × 224 images and applies agressive downsampling in the early layers. When applied to CIFAR-100's 32 × 32 images, important information is then lost and . We therefore make use of a commonly-used specialised ResNet architecture [Ide]

- Both a ResNet50 and a vision transformer on the Tiny Imagenet.

We summarize the datasets and models trained in the vision setting in We consider both the setting with a warmup followed by cosine decay learning rate schedule and constant learning rate.

Dataset	**Models**
MNIST, Fashion MNIST, SVHN	MLP, CNN
CIFAR10	Small ResNet (∼1M params)
CIFAR100	ResNet56[2], Densenet121
Tiny ImageNet	ResNet50, Vision Transformer

Table 4.1: Overview of datasets and model architectures used in experiments.

For both CIFAR-10/100, standard data augmentation techniques were implemented along with a tuned weight decay hyper-parameter.

In the case of pretraining language models, we restrict our attention to transformer based models. In particular we make use of the following enhanced implementation of nanoGPT ([Kar22]) including RMSNorm (instead of batch/layer normalization), SwiGLUi [Sha20] and Rotary Positional Embedding

For each model, we run a sweep over a range of learning rates with an optimally chosen weight decay and batch size. We run the learning rate sweep for both

Bias Correction isn't the End of the Story

When initially implementing the bias correction-free version of AdamW, we assumed that initialising the moments m_t, v_t with the gradients g_0, g_0^2 respectively would be an effective way to remove any bias - thus eliminating the need for bias correction. However, we discovered that this is not quite as simple as initially expected. While initialising the moments as zero certainly induces a bias, it is not the same bias that is corrected by bias correction.

We write down, in very explicit terms, the update step for the first and then all subsequent steps) in for each of the four possibilities (zero init, bias correction) ∈ (True, False) Consider the following closed form expression for the exponential moving averages:

$$m_t = \beta_1^t m_0 + (1 - \beta_1) \sum_{j=1}^{t} \beta_1^{t-j} g_j$$

We consider the four possible configurations of ZI and BC for the very first step of the optimizer:

---

[2] PyTorch's ResNet50 was designed for ImageNet's 224×224 images and applies aggressive downsampling in early layers. When applied to CIFAR-100's 32×32 images, important information is lost. We therefore use a specialized ResNet architecture [Ide].

g_1 is computed from the first batch passed in, then we have:

$$m_0 = \begin{cases} 0, & \text{if ZI} \\ g_1, & \text{else} \end{cases}$$

Then the EMA update occurs with the same grad (g_1 is used in first update too)
We have $m_1 = \beta_1 m_0 + (1 - \beta_1) g_1$

$$m_1 = \begin{cases} (1 - \beta_1) g_1, & \text{if ZI} \\ \beta_1 g_1 + (1 - \beta_1) g_1 = g_1, & \text{else} \end{cases}$$

Then bias correction is either applied or not. Therefore we have:

$$\hat{m}_1 = \begin{cases} g_1, & \text{if ZI and BC} \\ (1 - \beta_1) g_1, & \text{if ZI and no BC} \\ \dfrac{1}{1 - \beta_1} g_1, & \text{if no ZI and BC} \\ g_1, & \text{if no ZI and no BC} \end{cases}$$

By the very same logic, we have

$$\hat{v}_1 = \begin{cases} g_1^2, & \text{if ZI and BC} \\ (1 - \beta_2) g_1^2, & \text{if ZI and no BC} \\ \dfrac{1}{1 - \beta_2} g_1^2, & \text{if no ZI and BC} \\ g_1^2, & \text{if no ZI and no BC} \end{cases}$$

Thus the first optimizer step is given by (denote the unit vector in the direction of g_1 by $\hat{g}_1$ and these expressions are vectorized over all $g_1^{(i)}$):

$$\implies \text{first step} = s_1 = \frac{\hat{m}_1}{\sqrt{\hat{v}_1}} = \begin{cases} \dfrac{g_1}{\sqrt{g_1^2} + \varepsilon} = \dfrac{g_1}{|g_1| + \varepsilon} = \dfrac{1}{1 + \varepsilon / |g_1|} \text{sign}(g_1), & \text{if ZI and BC} \\ \dfrac{(1 - \beta_1) g_1}{\sqrt{1 - \beta_2}\, |g_1| + \varepsilon} = \dfrac{1 - \beta_1}{\sqrt{1 - \beta_2}} \dfrac{1}{1 + \frac{\varepsilon}{|g_1| \sqrt{1 - \beta_2}}} \text{sign}(g_1) & \text{if ZI and no BC} \\ \dfrac{\frac{1}{1 - \beta_1} g_1}{\frac{1}{\sqrt{1 - \beta_2}} |g_1| + \varepsilon} = \dfrac{\sqrt{1 - \beta_2}}{1 - \beta_1} \dfrac{1}{1 + \frac{\varepsilon \sqrt{1 - \beta_2}}{|g_1|}} \text{sign}(g_1), & \text{if no ZI and BC} \\ \dfrac{1}{1 + \varepsilon / |g_1|} \text{sign}(g_1), & \text{if no ZI and no BC} \end{cases}$$

The direction of the Adam update is then the same for all cases but the magnitude varies substantially. So the magnitude of the first gradient (which is basically random in some

capacity since the weights are randomly computed and there is just one noisy batch of gradient data to go by) basically determines the direction of the first step.
What matter is the order of magnitude of $|g_1|$ relative to For the case no ZI and BC, basically the factor $\frac{\sqrt{1-\beta_2}}{1-\beta_2}$ is making the gradient substantially larger (depends somewhat on values of β_1, β_2).

From the general formula:

$$m_t = \beta_1^t m_0 + (1-\beta_1)\sum_{j=1}^{t} \beta_1^{t-j} g_j \quad \text{and } \hat{m}_t = \frac{1}{1-\beta_1^t} m_t \text{ if BC else } m_t$$

and $(1-\beta_1^t) = (1-\beta_1)\sum_{j=1}^{t-1} \beta^j$

we consider the four cases with ZI and BC (let $B := \sum_{j=1}^{t-1} \beta_1^j$:

$$\hat{m}_t = \begin{cases} \frac{1}{B}\sum_{j=1}^{t} \beta_1^{t-j} g_j & \text{if ZI and BC} \\ (1-\beta_1)\sum_{j=1}^{t} \beta_1^{t-j} g_j & \text{if ZI and no BC} \\ \frac{\beta_1^t}{(1-\beta_1)B} g_1 + \frac{1}{B}\sum_{j=1}^{t} \beta_1^{t-j} g_j & \text{if no ZI and BC} \\ \beta_1^t g_1 + (1-\beta_1)\sum_{j=1}^{t} \beta_1^{t-j} g_j & \text{if no ZI and no BC} \end{cases}$$

why not just divide by $1-\beta_2$? rather than $1-\beta_2^t$? makes no difference up to scaling

4.3 Is Bias Correction just Hidden Learning rate scheduling?

In our experiments comparing adam with and without bias correction, we found that certain choices of the pair β_1, β_2 caused a much greater discrepency in performance, but only when no learning rate schedule was used. This lead to the following insight.

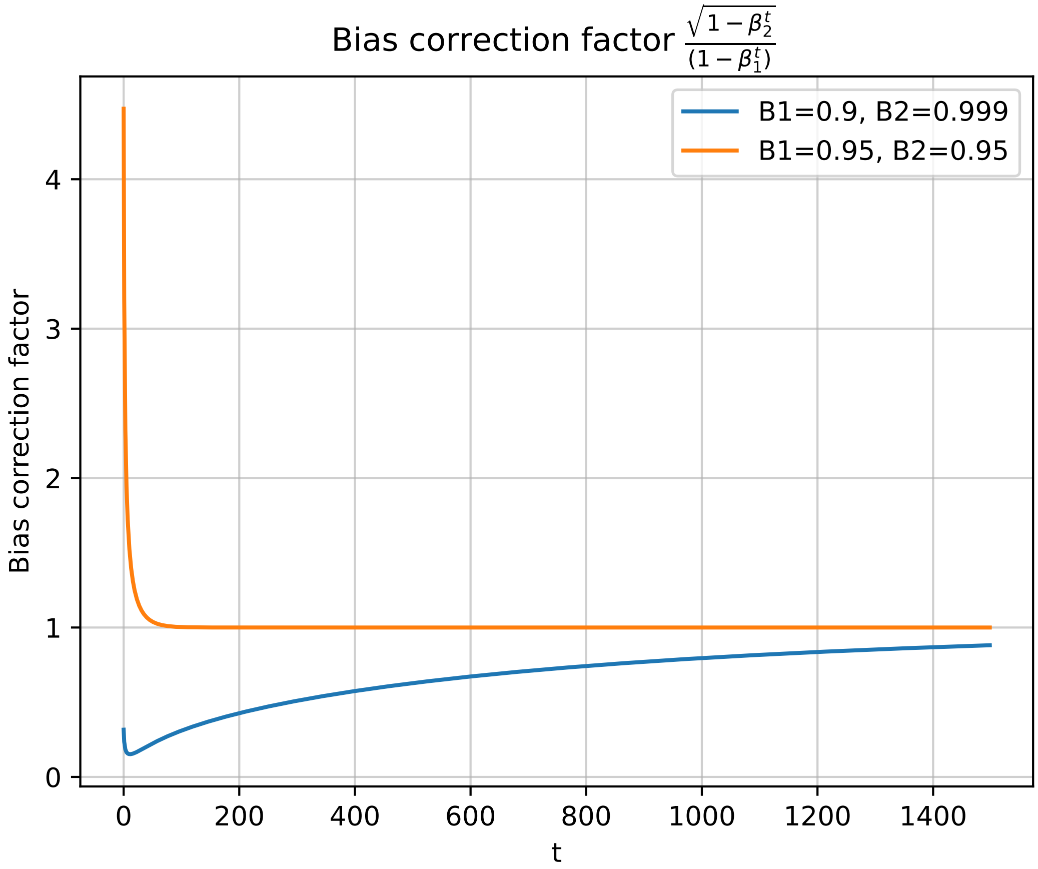
Consider the following factorization (we will ignore the ϵ in the denominator for simplicity) [3]

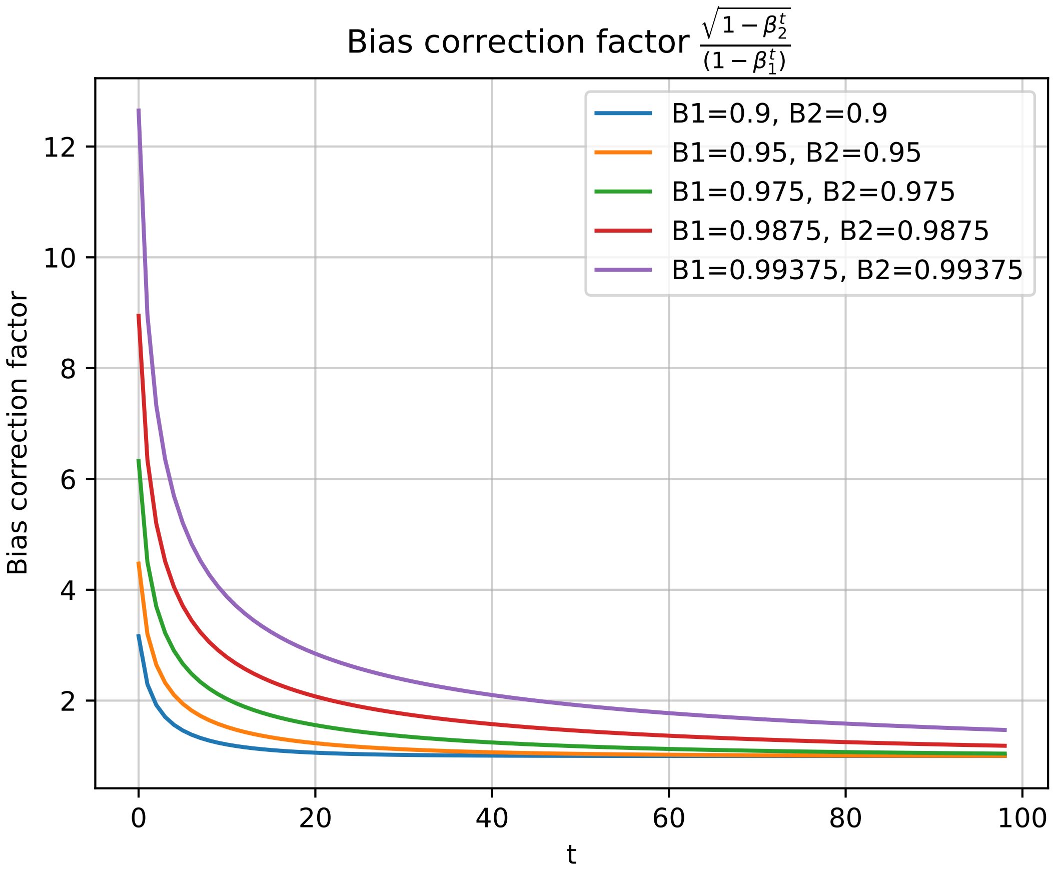
$$\frac{\hat{m}_t}{\hat{v}_t} = \frac{\frac{1}{1-\beta_1^t} m_t}{\sqrt{\frac{1}{1-\beta_2^t} v_t}} = \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}\frac{m_t}{\sqrt{v_t}} = \rho(t;\beta_1,\beta_2)\frac{m_t}{\sqrt{v_t}} \tag{4.1}$$

---

[3] We note that if the ϵ term is included, 4.1 evaluates to $\frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}\frac{m_t}{\sqrt{v_t}+\epsilon\sqrt{1-\beta_2^t}}$. Since $\sqrt{1-\beta_2^t} \in [0,1] \forall t \in \mathbb{N}\ \forall \beta_2 \in [0,1)$, bias correction should have minimal effect

The behaviour of $\rho(t;\beta_1,\beta_2) := \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$ which we refer to as the *bias-correction factor* depends highly dependent on the choice of $\beta_1, \beta_2 \in [0,1)$
Consider the following plot:





For the pair $(\beta_1, \beta_2) = (0.9, 0.999)$, the factor resembles a very steady warmup whereas for the pair $(\beta_1, \beta_2) = (0.95, 0.95)$, the factor it merely a large spike which quickly converges to 1

Consider the derivative of ρ with respect to t:

$$\frac{d\rho}{dt} = \frac{-\frac{\beta_2^t \log \beta_2 (1-\beta_1^t)}{2\sqrt{1-\beta_2^t}} + \sqrt{1-\beta_2^t}\beta_1^t \log \beta_1}{(1-\beta_1^t)^2}$$

where log denotes the natural logarithm.
Since $\log \beta_i < 0$ for any $\beta_i \in (0, 1)$, we note that ρ has positive t-derivative if and only if

$$\frac{\beta_2^t \log(\beta_2)}{2(1-\beta_2^t)} < \frac{\beta_1^t \log(\beta_1)}{1-\beta_1^t}$$

As we can see, in the case of $(\beta_1, \beta_2) = (0.95, 0.95)$, the derivative is *never* positive. It appears that what is far more important than the inclusion of bias correction is rather the initialization of the moments. One would naively assume that initializing the gradients to what they actually are should outperform setting them to zero. However there are minor improvements to initialising at zero

Table 4.2: *ZI* denotes Zero init, *BC* denotes Bias Correction. Not doing ZI means we initialize m and v at g_0 and g_0^2 respectively. Default for AdamW is ZI and BC. Performing bias correction is not as important as initialization in Adam. Averaged results over 4 random seeds
HPs lr:0.008, $\beta_1 : 0.95, \beta_2 : 0.95, wd : 0.1$

	AdamW	AdamW no BC, ZI	AdamW BC, no ZI	AdamW no BC, no ZI
Val ppl	21.87 ± 0.11	21.93 ± 0.04	22.83 ± 0.15	22.64 ± 0.13

Chapter 5

Beyond Adam: Is Muon the future?

While first-order optimization methods such as AdamW are currently the de-facto choice for training deep neural networks, In recent years, there have been a number of papers which attempt to improve upon Adam(W) by leveraging information about the structure of individual layers.
Newton's method, which requires explicit computation of the inverse-Hessian matrix, is rarely sufficiently efficient, or even tractable, in deep learning. Similarly, Natural Gradient Descent [Ama98] requires computation of the inverse-Fischer information matrix

Let $f : \mathcal{X} \times \Theta \to \mathcal{Y}$ with $\Theta \subset \mathbb{R}^p$ denote an arbirary deep neural network and let $L : \mathcal{X} \times \Theta \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ denote a (continuously differentiable) loss function.
The textbook definition of most traditional stochastic optimization algorithms involves initialising some $\theta_0 \in \Theta$ and then updating with some rule

$$\theta_{t+1} = \theta_t - \eta_t s_t$$

for some $s_t \in \mathbb{R}^d$ which is typically a function of the previous stochastic gradients $(g_j)_{j=1}^t$ where $g_j := \nabla_{\theta_j} L(x_j, \theta_j, y_j)$ and $(x_j, y_j) \subset \mathcal{X} \times \mathcal{Y}$ is the sampled mini-batch for each $j \in \{1, \ldots, t\}$.
Indeed, the vast majority of commonly-used first-order optimizers (SGD, RMSProp, Adadelta AdamW, Lion) ignore the structure of each individual layer of weights of a deep neural network and treat the parameters simply as a long concatenated vector made up of the flattened parameters of each layer.

An idea which has regained popularity in recent years is to exploit the individual structure of each layer of the matrix, leveraging this information to compute update directions tailored to each layer. For illustration's purpose, consider an MLP defined by

$$f($$

' The library Modula is built precisely on this notion, explicitely constructing a mathematical object called a *module* (not to be confused with the traditional interpretation in commutative algebra). These
A current SOTA language model is [LSY+25], which uses a mixture of experts (MoE)

transformer model architecture. The model explicitely uses Muon to optimizer linear (non-embedding) layers. Namely

5.1 Related Work

5.2 Understanding Muon

We provide a brief overview The following exposition builds on the work of [BN24a], [BN24b], [Ber25], [JJB+24], [PXA+25] supplemented with some additional pedagogy and proofs. **TODO: give a nice Muon explanation**

5.2.1 Metrized Deep Learning

5.2.2 Dualized Gradients and Metrized Deep Learning

The paper [BN24a] reviatalizes an old principle in optimization theory. Namely, the insistance on viewing the gradient as a vector living in the dual space to the corresponding weight space.

Consider a differentiable loss function $L : \Theta \to \mathbb{R}_{\geq 0}$ with respect to the weight space $\Theta \subseteq \mathbb{R}^d$. One can consider the first order Taylor approximation of L about arbitrary

$$L($$

5.2.3 Proof of Orthogonal Update step

5.2.4 Muon's Relationship to Shampoo

The Shampoo optimizer ([GKS18]) is intimately connected to the Muon update step.
Indeed

5.2.5 The Newton Schulz Algorithm

Under the framework of steepest descent under the RMS → RMS norm, the crucial step size for linear layers is seen as

5.3 Ablations on Simple Problems

To gain a better understanding of the dynamics of Muon, we study its behavior in simplified settings using both synthetic data and common toy datasets. Our goal is to examine how Muon compares to SGD and AdamW in these controlled environments, and to identify the minimal conditions under which Muon offers practical advantages.

The experiments were all run with the following codebase

5.3.1 Linear Regression

To whet our apetite, we begin by studying the case of linear regression.
Since Muon explicitely depends on the weight space that admits a matrix structure, we propose the following multi-output linear regression problem as the simplest for which Muon can be applied (methodology for synthetic data generation will then be discussed in detail later):

$$Y = XW + \varepsilon \quad \text{where: } \begin{cases} X \in \mathbb{R}^{N\times d} \text{ is our feature matrix} \\ W \in \mathbb{R}^{d\times D} \text{is our weight matrix }^{1} \\ Y \in \mathbb{R}^{N\times D} \text{ is our target space} \\ \varepsilon \in \mathbb{R}^{N\times D} \text{ noise where } \varepsilon_{i,j} \sim \mathcal{N}(0,\sigma_j^2) \text{ for } \sigma_1^2,\dots,\sigma_D^2 \in \mathbb{R}_{>0} \end{cases}$$

Indeed, excluding noise, this is an analogue of the linear portion of a hidden linear layer of a neural network. Here we are assuming no bias parameters. In any case Muon
The objective (or loss) function is then given by

$$L(W;X,Y) := \|Y - XW\|_F^2 \quad (*)$$

where $\|A\|_F := \sqrt{\sum_{i,j\in[m]\times[n]} |a_{ij}|^2}$ denotes the Frobenius norm of a matrix.
This objective function is manifestly convex in W. Indeed, we have

$$L(W) = \|Y - XW\|_F^2 = \|\text{vec(Y - XW)}\|_2^2 = \|\text{vec}(Y) - (\mathbb{1}_D \otimes X)\text{vec}(W)\|_2^2$$

jwhere $\otimes$ denotes the standard Kronecker Product and vec() denotes the vectorization of a rank k tensor into a rank 1 tensor.)
Clearly $(*)$ is simply a quadratic function of the form $f(z) = \|b - Az\|_2^2$ with positive-definite Hessian A^TA

Using elementary properties of the tensor product (namely that $(A\otimes B)^T = A^T \otimes B^T$ and $(A\otimes B)(P\otimes Q) := AP \otimes BQ$ whenever such a product makes sense), one can easily see that the identity:

$$(\mathbb{1}_D \otimes X)^T(\mathbb{1}_D \otimes X) = \mathbb{1}_D \otimes (X^TX)$$

The corresponding Hessian is therefore uniquely determined by the structure of the feature matrix X. While such a simple problem admits an analytic solution (indeed $W^* = (X^TX)^{-1}X^TY$ provided X^TX is invertible), the study of gradient-based optimizers on this problem can lead to valuable insights.

While such a simple problem admits an analytic solution (indeed $W^* = (X^TX)^{-1}X^TY$ provided X^TX is invertible), we study the case where the number of datapoints is large

We consider both the case of full gradient methods and stochastic gradient methods. The gradient of $(*)$ is given by

$$\begin{aligned}\nabla_W L &= \nabla_W \mathrm{tr}\left((Y - XW)^T (Y - XW)\right) \\ &= \nabla_W [\mathrm{tr}(Y^T Y) - 2\mathrm{tr}(W^T X^T Y) + \mathrm{tr}(W^T X^T XW)] \\ &= 2X^T (XW - Y)\end{aligned}$$

The stochastic gradient for minibatch $(X_t, Y_t) \subset \mathbb{R}^{N\times d} \times \mathbb{R}^{N\times D}$ $t \in \mathbb{N}$, which we denote by G_t, is given by

$$G_t = \nabla_{W_t} L(W_t) = 2X_t^T (X_t W_t - Y_t)$$

5.3.2 A Quadratic Problem

Another commonly studied and generally well understood problem in optimization is the quadratic problem.
In the simplest possible case, this problem is formulated as

$$\text{minimize}\left(\varphi(x) := \frac{1}{2} x^T A x\right) \tag{5.1}$$

Where $x \in \mathbb{R}^{m\times n}$ and $A \in \mathbb{R}^{m\times m}$ is usually a symmetric positive semidefinite.
Again, since Muon insists upon a weight space with a matrix structure, we reforumulate this problem into it's simplest matrix analogue:

$$\text{minimize}\left(q(W) := \frac{1}{2}\mathrm{trace}(W^T Q W) \equiv \|Q^{\frac{1}{2}} W\|_F^2\right) \tag{5.2}$$

where $W \in \mathbb{R}^{m\times n}$ and $Q \in \mathbb{R}^{m\times m}$
We note that again vectorization reduces (5.2) to a problem of the form of (5.1):

$$\begin{aligned}q(W) &= \frac{1}{2}\sum_{i=1}^{n} (W^T Q W)_{ii} \\ &= \frac{1}{2}\sum_{i=1}^{n} w_i^T Q w_i \quad (\text{where } w_i := \mathrm{col}_i(W)) \\ &= \frac{1}{2}\mathrm{vec}(W)^T (\mathbb{1}_n \otimes Q)\mathrm{vec}(W)\end{aligned}$$

TODO: Discuss a bit with antonio about the whole connection of these two problems, is parameter structure even a proper thing? Also the homogenous vs heterogeneous Hessian situation.
Provided the matrix

5.3.3 Logistic Regression

Similarly to the linear regression case, we consider multi-class classification with a softmax function. Again, because we are interested in the Muon optimizer, this is the simplest

case for which the weights admit a natural matrix structure.
One can view Logistic regression as a neural network of just one layer so it is certainly worthy of scrutiny.

- Generate X and W^* as in the linear regression case
- Let $Z = XW^*$ and $P = \text{softmax}(Z) + \varepsilon$ for noise ε
- Define $Y_{ik} := \begin{cases} 1, & \text{if } k = \text{argmax}_{\ell \in \{1,\dots,K\}} P_{i\ell} \\ 0, & \text{else} \end{cases}$ for each $i \in \{1, \dots, N\}$ (or in plain english: each row of Y is a one hot encoded vector in $\mathbb{R}^K$)
- Initialise a random $W \in \mathbb{R}^{(d+1)\times D}$ and compare different optimizer's performance on minimizing loss.

Let $f(X; W) := \text{softmax}(XW)$ As is standard for classification, we consider the cross-entropy loss:

$$L(W) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} Y_{ik} \log(P_{ik}) \quad \text{where } P = \text{softmax}(XW)$$

Let us compute the gradient. We denote $P = \text{softmax}(Z)$ where $Z := XW$. We invoke the chain rule to simplify matters:

$$\nabla_W L = \nabla_Z L \,.\nabla_W Z = \frac{1}{N} X^T (\text{softmax}(XW) - Y)$$

5.3.4 Small MLPs

5.4 SVD structural analysis

The Muon optimizer makes use of the Newton Schulz algorithm

5.4.1 CIFAR-10 ResNet

The CIFAR speedrun is a competitive benchmark which aims to achieve a specified test set accuracy in the shortest possible training time (standardized in NVIDIA A100-minutes). Researchers submit their optimizations, which can include network architecture changes, alterations to the training pipeline and test time tricks, and the submission is then benchmarked by averaging over a number of random seeds. The goal is to achieve the shortest average training time while exceeding a pre defined test set accuracy.
Remarkably, with enhancements using Muon (**TODO: cite the right thing**), the current record is a mere 2.59 seconds to achieve 94% training accuracy.
We make use of this repository to ablate on some hyperparameter choices for the Muon optimizer while enjoying minimal experiment runtimes.

Comparing Newton Schulz approximation to full Gradient Orthogonalization

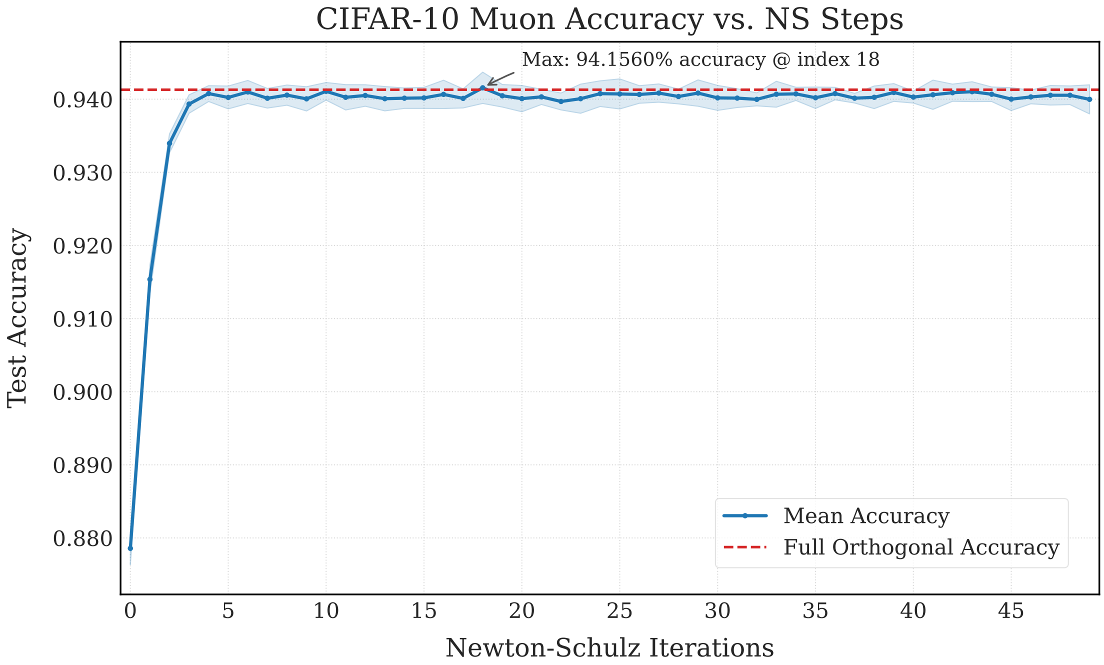


Figure 5.1: Ablation varying number of Newton Schulz steps performed and comparison with performance with an exact orthogonalization computed using the SVD of the matrix. Each data point on the x-axis corresponds to an averaged final test accuracy over 20 random seeds with error bars representing a 95% confidence interval. Other hyperparameters are as in record speedrum configuration.

5.5 Experimental

Chapter 6

Discussion and Future Work

Bibliography

[Ama98] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[Ber25] Jeremy Bernstein. Deriving muon, 2025.

[BN24a] Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning, 2024.

[BN24b] Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology, 2024.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[GKS18] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization, 2018.

[HLvdMW18] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[Ide] Yerlan Idelbayev. Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch. https://github.com/akamaster/pytorch_resnet_cifar10. Accessed: 20xx-xx-xx.

[JJB$^+$24] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024.

[Kar22] Andrej Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT, 2022.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[LSY+25] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for llm training, 2025.

[PXA+25] Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained lmos, 2025.

[Sha20] Noam Shazeer. GLU variants improve transformer. *CoRR*, abs/2002.05202, 2020.

[YG20] Wei Yuan and Kai-Xin Gao. Eadam optimizer: How ϵ impact adam, 2020.

Appendix A

How Exponential Moving Averages Smooth the Update Signal

It is generally well accepted that