Thesis Notepad of Ideas and Progress etc

June 12, 2025

Contents

A closed form for weight decay updates

Just something i was playing with: a derivation for a closed form expression for θ^{k+1} in terms of the initialised weights θ_0 and the step updates $(s_j)_{j\in 0:k}$

Suppose we are dealing with a weight decay update step of the form

$$w_{k+1} = (1 - \eta_k \alpha w_k) + \eta_k s_k$$

where $\alpha \in \mathbb{R}_{>0}$ is the weight decay parameter, $(\eta_k)_k$ is the learning rate scheduler and $(s_j)_j$ are the update step sizes. (i.e. $s_k = \frac{m_k}{\sqrt{v_k}+\varepsilon}$ for AdamW or $s_k = g_k$)
Then we have

$$\begin{aligned} w_{k+1} &= (1 - \alpha\eta_k)w_k - \eta_k s_k \\ &= (1 - \alpha\eta_k)[(1 - \alpha\eta_{k-1})w_{k-1} - \eta_{k-1}s_{k-1}] - \eta_k s_k \\ &= (1 - \alpha\eta_k)(1 - \alpha\eta_{k-1})[(1 - \alpha\eta_{k-2})w_{k-2} - \eta_{k-2}s_{k-2}] - \eta_{k-1}(1 - \alpha\eta_k)s_k - \eta_k s_k \end{aligned}$$

By induction, we obtain the formula

$$w_{k+1} = P_k w_0 - \sum_{j=0}^{k} \eta_{k-j} P_j s_j, \quad P_r := \begin{cases} 1, & \text{if } r = 0 \\ \prod_{j=0}^{r}(1 - \alpha\eta_{k-j}), & \text{if } r \geq 1 \end{cases}$$

As shown in the paper on Constrained parameter Regularization (link to paper) one can view α as a vector in $\mathbb{R}^d$ where we subdivide α into several blocks so we can regularize different groups of parameters of the network to different degrees. Namely we can also view $\alpha = (\alpha_1, \ldots, \alpha_p)$ for sub-collections of parameters $\alpha_j \subset \alpha \in \mathbb{R}^d$.
Consider the relation

$$P_k = P_i \prod_{i=j+1}^{k} (1 - \alpha\eta_i) \implies P_j = \frac{P_k}{\prod_{i=j+1}^{k}(1 - \alpha\eta_i)}$$

Then we can write

$$w_{k+1} = P_k \left(w_0 - \sum_{j=0}^{k} \frac{\eta_{k-j}}{\prod_{i=j+1}^{k}(1 - \alpha\eta_i)} s_j \right)$$

Look at some kind of $\log()$, expectation, variance etc and investigate progression.
Also interesting to see the angle/ inner product of w_k vs w_0 to see how things progress with the weight decay

Getting the following convergence guarantee:

$$\begin{aligned} E\left[\|w_{k+1} - w_0\|\right] &= E\left[\left\|(1 - P_k)w_0 + \sum_{j=0}^{k} \eta_{k-j} P_j s_j\right\|\right] \\ &\leq |1 - P_k| E[\|w_0]\| + \sum_{j=0}^{k} \eta_{k-j} P_j E[\|s_j\|] \\ &= \sum_{j=0}^{k} \eta_{k-j} P_j E[\|s_j\|] \quad \text{(since the first term should be 0)} \end{aligned}$$

The expected value of the norm of w_0 should be 0 in most initialization schemes

Chapter 1

Introduction

Chapter 2

Paper Review

2.1 Weight decay papers

Perspectives on Weight Decay for LLMs

It is absolute standard practice to implement weight decay when training almost any SOTA deep network. Yet despite its ubiquity, its role in training dynamics in not yet completely understood.
A number of recently published papers attempt to understand the role of weight decay. We provide a brief summary of each paper before discussing some possible ideas when using adaptive optimizers.

Why Do We Need Weight Decay in Modern Deep Learning?

link to paper
In a nutshell, this paper analyses the role weight decay plays in *over-training* and *under-training* regimes. In the case of overtraining regimes (such as ResNet or the majority of vision tasks), the large number of passes through the data necessitates the employment of regularization to prevent overfitting. Weight decay, in this case, is therefore employed and serves as a fairly standard means to prevent overfitting. However as shown in Zhang et al (2016), even with strong weight decay, such overtrained networks can fully memorize the data. The authors then study vision tasks trained on SGD and show how the optimization dynamics are modified by the very presence of weight decay and the implicit regularization of SGD.

We are, however, more interested in the under-training regime since this is what the training of large language models falls into. Due to the large quantity of training data, much fewer passes through the training data are required (or indeed even possible).

Improving Deep Learning Optimization through Constrained Parameter Regularization

link to paper

This paper introduces an interesting adjustment to the traditional weight decay paradigm. Rather than imposing a single weight decay parameter $\gamma \in \mathbb{R}_{>0}$, several different weight decay constants γ_j are defined for sub collections of parameters $\theta_j \subseteq \theta$. The idea here is to prevent rigidity and allowing for regularization to different extents to different parameter matrices.

The authors also phrase the problem in the form of a constrained optimization problem and perform the weight updates with respect to the solution of this optimization problem.

TO BE COMPLETED

Chapter 3

A Brief History of Optimization

Chapter 4

Understanding Adam

We discuss some of the underlying theory, review some important literature and run some experiments with the aim of understanding the dynamics of AdamW and the role which each hyperparameter plays in Adam's efficacy.
Experiments

4.1 Why is AdamW the Top Dog?

The Adam optimizer [7] was introduced in 2017. With the addition of decoupled weight decay [9], it has been the de facto stochastic optimizer choice for nearly all SOTA deep learning training, particularly in language model pretraining. Extensive research into understanding Adam's dynamics has been conducted

The optimizer itself is merely a synthesis of two already well-established optimization ideas - momentum and RMSProp. Adam enjoys the upsides of both methods and has proved to be a powerful and robust choice
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.
While a substantial number of publications claim great and consistent

4.2 Preliminaries and Related Work

The AdamW optimizer can be represented as follows:

4.2.1 EAdam Paper probably

4.2.2 Relation to signSGD

4.2.3

4.3 Some Experiments/Ablations

Need a lot of sensitivity curves for models in vision and language setting.

- 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.

4.3.1 How EMA smooths the update signal

4.3.2 The ε Hyperparameter

While regular adam is given by:

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

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

It has been observed in a number of papers ([12]), 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

4.3.3 Nesting the Moving averages

Coupling of momentum and RMS

The momentum and RMS boil down to keeping track of a moving average (EMT_β) parametrized by $\beta \in \mathbb{R}_{\geq 0}$.

While regular adam is given by:

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

one could also consider the double nesting formula

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

Or expressed in two steps:

$$v^t = \beta_2 v^{t-1} + (1-\beta_2) g^t \odot g^t$$
$$K^t = \beta_1 K^{t-1} + (1-\beta_1) \frac{g_t}{\sqrt{v^t} + \varepsilon}$$

Of course, the
When doing this, the bias correction term becomes less clear cut. Of course the second moment estimate v^t is the same ($1-\beta_1^t$ if you believe $E[g_t]$ is similar on initialization) . But after the nested average we have a sum of the form

$$K^t = (1-\beta_1) \sum_{j=0}^{t-1} \beta_1^{t-i-1} \frac{g_j}{\sqrt{v_j} + \varepsilon}$$

Can we make the same kind of assumption for initial values of t? Namely that $E[\frac{g_j}{\sqrt{v_j}+\varepsilon}]$ is "close enough" for all small j so that we can reduce the ugly expression above to a geometric sum. If so, great the term is the same. If not, any other clever tricks?
Indeed as we demonstrate in detail in 5 that bias correction is currently poorly understood and is generally not required if proper learning rate scheduling is implemented.

Chapter 5

Bias Correction in Adam: A Detailed Analysis

The AdamW optimizer can be represented as follows:(need to replace this with the algorithm itself for sure)

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

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.
One observes in a large number of research papers (gotta find some nice examples), an extensive discussion into the role of bias correction. 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.

A Discussion of the "Proof" for Bias Correction

In a number of pedagogical resources (cite Goodfellow, Geiger lecture series), blogs and indeed seminal research papers - some variant of the following "proof" is given to justify

the inclusion of bias correction:

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

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 patently false. We argue through a number of experiments with a custom implementation of AdamW that the bias correction step can be removed from AdamW with no adverse effects.

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) $\in$ (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:
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

5.1 Is Bias Correction just Hidden Learning rate scheduling?

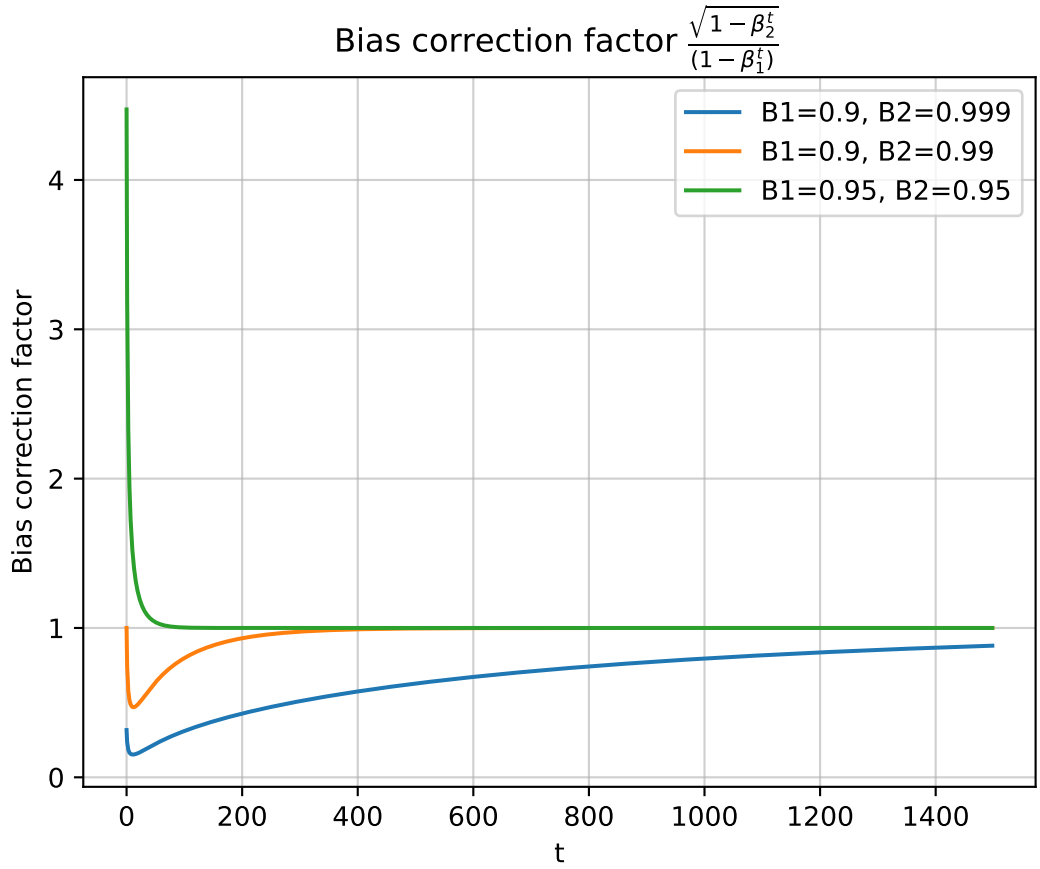
In our experiments comparing adam with and without bias correction, we found that certain choices of the pair β_1, β_2 caused a much greater discrepency in performance, but only when no learning rate schedule is used. This lead to the insight we present below.

Consider the following factorization (we will ignore the ε in the denominator for simplicity [1]

$$\frac{\hat{m}_t}{\hat{v}_t} = \frac{\frac{1}{1-\beta_1^t}m_t}{\sqrt{\frac{1}{1-\beta_2^t}v_t}} = \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}\frac{m_t}{\sqrt{v_t}} \tag{5.1}$$

he behaviour of this factor $\rho(t;\beta_1,\beta_2) := \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$ depends greatly on the choice of $\beta_1, \beta_2 \in [0,1)$
Consider the following plot:

[1] We note that with ε included, 5.1 evaluates to $\frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}\frac{m_t}{\sqrt{v_t}}$



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.

Experimental Outline

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

- A less overparametrized ResNet implementation on the CIFAR10 dataset (only $\sim$ 1M parameters rather than ResNet 18 which has $\sim$ 11M)
- Both a ResNet56[2] and a vision transformer on the CIFAR100 dataset
- Both a ResNet50 and a vision transformer on the Tiny Imagenet and Imagenet datsets

We consider both the pre-training and fine-tuning setting for CIFAR100/TinyImagenet and just the pretraining setting for CIFAR10.

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 ([6]) including RMSNorm (instead of batch/layer normalization), SwiGLUi [11] and Rotary Positional Embedding
We note that the exact hyperparemeter configuration used for each experiment is not of integral importance in distinguishing the performance of AdamW with and without bias correction (and indeed zero-initialistion). In each case, we run a sweep to ensure a model of near SOTA performance but this is mainly for vanity and we also consider results across a wide range of hyperparameters to prove the performance similarities are consistent across the space of all hyperparameter configurations.

Table 5.1: *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

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

[2]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 [4]

Chapter 6

Beyond Adam: Is Muon the future?

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 Hessian matrix, is rarely sufficiently efficient, or even tractable, in deep learning. Similarly, Natural Gradient Descent [1] makes use of the Fischer information matrix
A general principle of stochastic optimizers in the age of deep learning is to approximate local information about the curvature of the loss landscape without needing to explicitely compute the full second derivative. Knowledge of local curvature is known to help prevent becoming stuck in bad local minima.

Let $f : \mathcal{X} \times \Theta \to \mathcal{Y}$ with $\Theta \subset \mathbb{R}^p$ denote an arbirary deep neural . The textbook definition of most traditional stochastic optimization algorithms involves initialising $\theta_0 \in \Theta$ and then updating with some rule

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

for a step size $s_t \in \mathbb{R}^d$ which typically depends on the current stochastic gradient.
The vast majority of commonly-used first-order optimizers (e.g. 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.
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 [8], which uses a mixture of experts (MoE) transformer model architecture. The model explicitely uses Muon to optimizer linear (non-embedding) layers. Namely

6.1 Related Work

6.2 Understanding Muon

The idea of Muon stemmed from (**TODO: cite the modular norm papers from bernstein**) The following exposition builds on the work of [3], [2], [5], [10] supplemented with some additional pedagogy and proofs. **TODO: give a nice Muon explanation**

6.2.1 Metrized Deep Learning

6.2.2 Dualized Gradients and Metrized Deep Learning

The paper [3] reviatalizes an old principle in optimization theory. Namely, the insistance on viewing the gradient as a dual vector.
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

$$L($$

6.2.3 The Newton Schulz Algorithm

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

6.3 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

6.3.1 Linear Regression

We consider the case of linear regression with a multi-dimensional output. Namely, consider a datasetThis is the simplest setting in which Muon can be applied as the weights for the one-dimensional output case would not admit a matrix structure. Here the dynamics of stochastic optimizers are generally well understood. We strive to understand how Muon behaves in this setting and how it differs from SGD and AdamW. We generate a synthetic dataset as follows:

- Generate data $Z \in \mathbb{R}^{Nxd}$ with $z_{ij} \sim \mathcal{N}(0, 1)$ (corresponding to N datapoints $x_i \in \mathbb{R}^d$) and let $X = [1|Z] \in \mathbb{R}^{d+1}$ (maybe generate better data than just normal)

- Generate a weight matrix $W^* \in \mathbb{R}^{d\times D}$ from a unit Gaussian $\mathcal{N}(0,1)$
- Generate noise $\varepsilon \in \mathbb{R}^{N\times D}$ with $\varepsilon_{ij} \sim e\mathcal{N}(0,1)$ for hyperparameter $e \in \mathbb{R}_{>0}$
- Let $Y = XW^* + \varepsilon$ and initialise $W \in \mathbb{R}^{d\times D}$

The objective (or loss) function is then given by

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

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

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

which is simply a quadratic function of the form $f(x) = \|y - Ax\|_2^2$ with positive-definite Hessian $2A^TA$.
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 experiment with both mini-batch and full gradient methods.
The gradient of $(*)$ is given by

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

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

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

We consider the singular value decomposition (SVD) $X = U\Sigma V^T$ of the feature matrix. Then the gradient G (up to a constant) is given by:

$$G = X^T(XW - Y) = V\Sigma^TU^T(U\Sigma V^TW - Y) = V^T\Sigma^T\Sigma\, VW - V\Sigma^TUY$$

The analytic solution (obtained at $G = 0$ due to convexity) is given by:

$$(X^TX)^{-1}X^TY = V\Sigma^{-2}V^TV\Sigma U^TY = V\Sigma^{-1}U^TY$$

Since the Muon update dictates considering $\text{NewtonSchulz}_n(G_t)$ which is just an approximation of the orthogonalization of G_t, we will study the singular value decomposition of G at each time step and study how a variety of factors
It is well established that the condition number of the feature matrix is a good indicator of the problem's numerical stability and learning difficulty. Therefore we are interested in the interplay between

6.3.2 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:

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

6.4 SVD structural analysis

The Muon optimizer makes use of the Newton Schulz algorithm

6.4.1 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 over a number of random seeds. Remarkably, with enhancements using Muon (**TODO: cite the right thing**), the 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.

Comparing Newton Schulz approximation to full Gradient Orthogonalization

6.5 Muon on nanoGPT and plainLM

Bibliography

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

[2] Jeremy Bernstein. Deriving muon, 2025.

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

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

[5] 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.

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

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

[8] 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.

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

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

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

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