Thesis Notepad of Ideas and Progress etc

May 16, 2025

Contents

Chapter 1

Math formulation of ideas

1.1 Adam Ideas

The Adam algorithm is applied in the following way.
For each iteration $t \in \mathbb{N}$, let $g^t := \nabla_w L(w)$. The update step is given by

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

LR scheduler ideas

It is sometimes the case that a learning rate scheduler $(\eta_j)_{j\in\mathbb{N}}$ is used in conjunction with Adam. One could also encode a decreasing (resp. increasing) step size by changing the ε parameter with time.
Namely, one has update steps of the form

$$w^{t+1} = w^t - \eta \frac{\hat{m}^{t+1}}{\sqrt{\hat{v}^{t+1}} + \varepsilon_t} \quad \text{instead of} \quad w^{t+1} = w^t - \eta_t \frac{\hat{m}^{t+1}}{\sqrt{\hat{v}^{t+1}} + \varepsilon}$$

probably stupid

Easiest way to alter the existing adam method in torch is just to find the equivalent learning rate schedule which corresponds to a given ε schedule.
More explicitely, given some sequence $(\varepsilon_t)_{t\geq 0}$ and $\eta > 0$, what sequence $(\eta_t)_{t\geq 0}$ is such that

$$w^{t+1} = w^t - \eta \frac{\hat{m}^{t+1}}{\sqrt{\hat{v}^{t+1}} + \varepsilon_t} \quad \text{is equialent to} \quad w^{t+1} = w^t - \eta_t \frac{\hat{m}^{t+1}}{\sqrt{\hat{v}^{t+1}} + \varepsilon_0}$$

Solving would give:

$$\eta_t = \frac{(\eta - \eta_t)\sqrt{\text{EMA}_{\beta_2}(g_t^2)} + \eta\varepsilon_0}{\varepsilon_t}$$

Ok definitely stupid: can't know lr ahead of time. Will just implement directly in torch.

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 in two steps

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

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?

to also consider

- The Frederic Kunstner paper discusses Zipf's law and looking at loss of less common tokens. Shows adam learns the less frequent tokens better than SGD even with no batches
- the Bernstein paper shows how (without first moment ema) most optimizers (e.g Adam, SGD, Prodigy) are simply steepest descent wrt to some norm and how different parameters should have different operator norms applied depending on shape and action. Adam chooses infinity norm and kind of so does SignSGD

- the paper from 2020 "geometry of sign GD" shows that maybe sign methods can sometimes be better. In particular when Hessian is centred near diagonal and when the max eigenvalues are much bigger than average eigenvalues. Also SGD and Adam have some similaraties and are equivalent as params in adam $\to 0$. Also ℓ_∞ smoothness is weaker than seperable smoothness and is a sufficient assumption.
- Maybe some discussion on Music Foundation models?
- implementation pytorch-optimizer forked repo. Good idea?

Week of 25th November

- Read the Optimization for deep learning: Theory and algorithms document
- implemented the ε-scheduler to optimizer (torch optimizer prototypes forked from online repo)
- read the confidential paper sent via slack.
- read Dynamics of SGD with Stochastic Polyak Stepsizes: Truly Adaptive Variants and Convergence to Exact Solution
- Began benchmarking Adam variants on some toy datasets in collab: the cluster contract still awaits!
- Looked at Nikkolo's PlainLM repo repo
- tried reading up a bit on best practices of hyperparameter sweeps etc. Some basic questions

Week of 2 December

- Read A Survey on Efficient Training of Transformers
  - one proposition is SOAP without accumulation for efficiency
  - AdaDiag basically uses an SVD of the gradient to precondition the step size which still doing ADAM type things. Might be an interesting thing to try and nest with the NestedMA optimizer.
- IMPROVING ADAPTIVE MOMENT OPTIMIZATION VIA PRECONDITIONER DIAGONALIZATION
- investigating the modula package (Scalable Optimization in the Modular Norm)
- Looked over the AdamMini paper
- looked over Why transformers need Adam: A Hessian Perspective

some thoughts

- would be interesting to investigate bias correction vs taking some steps with SGD and then initializating adaptive optimizer from there
- would like to try the bernstein norm and explictely assign a different optimizer paradigm depending on the type of layer as discussed in the paper "Scalable Optimization in the Modular Norm".
- What are some reasonable ways for the epsilon scheduler to update. Intuitively a milestones = [] seems useful but maybe a functional relationship could be better? generally ε should increase to have something of a similar affect to the learning rate shrinking but to which extent? is there some theory to be built?
- evalute on vision transformers and small language models, RNNs, GNNs etc. to start to see if adaptive optimizers are more a data or model choice or both...
- Adam usually updates the moving averages per batch but it might be interesting to update more strongly for recent batches since if we use mini batching very quickly we lose info within a single epoch even though it might be nice to retain it

Week of 9 December

- The Impact of the Mini-batch Size on the Variance of Gradients in Stochastic Gradient Descent
- Which Algorithmic Choices Matter at Which Batch
- Why Do We Need Weight Decayin Modern Deep Learning?
- AdAdaGrad: Adaptive Batch Size Schemes for Adaptive Gradient Methods
- Adam mini
- this muon thing

ideas

- Work with some adaptive batch size; in the paper AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks, they implement for vision models. They also call it "adaptive" but really just make a batch size schedule and then make sure the ratio of the learning rate to the batch size remains constant. Also not clear whether they look at the adaptive adam step size or just the fixed learning rate (get the impression it's the latter). Don't see many other papers on this topic and the takeaway of the paper was that performace is similar but since the batch size increases, the later epochs are quicker and therefore training time is reduced.

- This backPACK package let's us compute the variance of batches (or could just do as in The Impact of the Mini-batch Size on the Variance of Gradients in Stochastic Gradient Descent).
  Idea to monitor variance as proxy for batch update size
  At iteration t:
  Obtain g_t and $Varg_t$
  Define controller which takes in the batch size, learning rate, epoch and gradient and controls how large the variance is.

  Techniques like Inner Product test,

- Which Algorithmic Choices Matter at Which Batch Sizes? Insights From a Noisy Quadratic Model

- get partial hessian of some of the diagonal blocks (the hessian perspective on adam paper shows the structure is basically block diagonal in later epochs). Block diagonal hessian free method kind of ignores the curvature entirely

- other network architectures

1.2 January 2025

idea

Let's consider a harmonic mean of the step size in SGD and the step size in Adam. Note that up to rescaling, the 2 in the harmonic mean formula can be eliminated.

$$\text{statistic} = \frac{1}{\frac{1}{(\frac{\text{EMA}_{\beta_1}(g_k)}{\sqrt{\text{EMA}_{\beta_2}(g_k}})+\frac{1}{\eta_k}}} = \frac{1}{\frac{\sqrt{\text{EMA}_{\beta_2}}(g_k)}{\text{EMA}_{\beta_1}(g_k)}+\frac{1}{\eta_k}}$$

The $\frac{1}{\eta_k}$ can be seen as an ε_k parameter in some sense.
However this doesn't seem to extract precisely what we want which is for adam to exactly fall out with an $\varepsilon_k \iff \frac{1}{\eta_k}$ correspondence. It therefore makes sense to maybe search for a step size $s(\beta_1, \beta_2, k)$ for which

$$\frac{1}{\frac{1}{s}+\frac{1}{\eta_k}} = \text{Adam step}$$

idea

Investigate the failing of using statistics other than the mean in backpropagation. BackPACK provides access to variance for basic networks. However it never seems as though having access to variance actually assists us in making better step sizes. Have begun to design an experiment in which we get the variance of the gradients and the mean each epoch and then compare this to the EMA seen in Adam. Claim that the v_t^2 is simply modelling the 2nd moment even though it moving

average of exponenets of a sample mean.
More explicitely, if denote $G_t = [g_{t,1}, \ldots, g_{t,B}]$ so that $g_t = \text{mean}(G_t, \text{dim} = 1)$ for each t, we have

$$v_t^2 = \text{EMA}_{\beta_2}(g_t^2) = \text{EMA}_{\beta_2}\left(\left(\frac{1}{B}\sum_{b=1}^{B} g_{t,b}\right)^2\right) = \text{EMA}_{\beta_2}\left(\frac{1}{B^2}\sum_{b,c} g_{t,b}g_{t,c}\right)$$

It would be interesting to see if as $t \to \infty$, does the distribution parametrized by m_t and v_t approach the same distirubtion as parametrized by the variance and mean at epoch t.
Idea to compute the KL divergence of normals between $(g_t, \text{diag}(\text{Variance}))$ and $(m_t, \text{diag}(v_t - m_t^2)$ and see whether the divergence remains the same, approaches zero, or something else.
Closed form for two KL divergences is

$$D_{KL}(p \,\|\, q) = \frac{1}{2}\left[\text{tr}(\Sigma_q^{-1}\Sigma_p) + (\mu_q - \mu_p)^\top \Sigma_q^{-1}(\mu_q - \mu_p) - k + \ln\left(\frac{\det \Sigma_q}{\det \Sigma_p}\right)\right]$$

made particularly simple where the covariance matrices are diagonal.

idea

Understand the distribution of gradients for Adam by plotting the mean and deviation for various dimensions as the iteration grows
maybe understanding just particular blocks is sufficient

idea

Investigate how signSGD with very large batch sizes works in this paradigm since intuitively the variance should be very low in the case of large batches.

plainLM

Forked Niccolo's plainLM repo (minimalLM)
Having some issues and talking to Niccolo tomorrow but will soon be able to run experiments. Namely have the Nested moving average adam variant with an epsilon scheduler.

more papers

- weight decay talk from lousanne
- D'Angelo et al paper. Interesting Bias-Variance tradeoff relation. Not really functioning as a true regularizer but changing training dynamics.
- gradient norm perspective on weight decay. Interesting but not really talking about language models.
- after reading Hennig paper connecting signSGD and Adam, looked through some more sign sgd papers but nothing too interesting.

Epsilon Scheduler stuff

Let $(\eta_k)_{k\geq 1} \subset \mathbb{R}_{>0}$ be a monotone decreasing learning rate scheduler for an Adam learning rate scheduler (if using a warmup learning rate schedule this won't apply until after warmup is complete).

Then to find the choice of a scheduled series $(\varepsilon_k)_{k\geq 1}$ which aligns to using the sequence $(\eta_k)_{k\geq 1}$, we can solve the equation

$$\eta_k \frac{m^k}{\sqrt{v^k}+\varepsilon} = \eta \frac{m^k}{\sqrt{v^k}+\varepsilon_k}$$

where we understand $\eta := \eta_0$ and $\varepsilon := \varepsilon_0$
We solve this to give

$$\varepsilon_k = \frac{(\eta-\eta_k)}{\eta_k}\sqrt{v^k} + \frac{\eta}{\eta_k}\varepsilon$$

i.e. the corresponding epsilon sequence is given by a weighted sum of the ε and $\sqrt{v^k}$ terms (where ε is understood to be broadcasted to $\dim(g_t)$

For instance, the adam-mini library divides into blocks. Makes sense to try using several different ε for blocks like the way adam-mini used several global second moments for a corresponding block.
Also want to generally monitor the $\sqrt{v_k}$ values (maybe with a norm, min, max) and see how it changes.

Proposed scheduler

We design a scheduler of the following form. Denote by $T \in \mathbb{N}$ the total number of steps to be taken.

- For the first wT steps($w \in [0,1)$), use a small ε_0 initial value ($\varepsilon = 1e-8$)
- Then for all steps $t \in [wT, ET)$ ($E \in (w,1)$) apply exponential growth to the epsilon value controlled by some growth facter $\kappa \in \mathbb{R}_{>0}$
- For all steps $t \in [ET, T]$ apply a max value which is akin to making the step size SGD

The idea is to somehow smoothly transition from classic AdamW to SGD since often training some steps of SGD at the end of training is beneficial.
Let $\varepsilon_0, \varepsilon_{\max}$ be the initial and final epsilon values The formula for $\varepsilon(t)$ is of the form $\varepsilon(t) = Ae^{k(t-wT)} + B$. Solving for A,B gives

$$A = \frac{\varepsilon_{\max}-\varepsilon_0}{e^{kt(E-w)}-1} \quad , \quad B = \varepsilon_0 - A$$

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 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

Some Kind of Plan/Structure

Generally looking at adamw and related adaptive optimizers and investigating several of the factors which make it a supreme optimizer

- Introduction to outline idea
- Background Reading/Recent papers of interest for inspiration
- Chapter about bias correction and how it's not actually needed (maybe just part of background)
- analysis of some other optimizers. SGD $\rightarrow$ AdamW $\rightarrow$ Shampoo, SOAP, Muon
- Adam Moments/KL divergence
- Is it the data or the Transformer architecture that needs adam (experiment with ViT on imagenet and LLM)
- nested MA optimizer
- epsilon scheduler and using blockwise updates rather than same for every parameter
-

I don't understand the SGD implementation at all :////

I was under the impression Adam with $\beta_2 = 0$ was simply sgd with momentum. However there are several key differences in the implementation

The step implementation in SGD looks something like

```
for i, param in enumerate(params):
    grad = grads[i] if not maximize else -grads[i]
    if weight_decay != 0:
        grad = grad.add(param, alpha=weight_decay)
    if momentum != 0:
        buf = momentum_buffer_list[i]
        if buf is None:
            buf = torch.clone(grad).detach()
            momentum_buffer_list[i] = buf
        else:
            buf.mul_(momentum).add_(grad, alpha=1 - dampening)
        if nesterov:
            grad = grad.add(buf, alpha=momentum)
        else:
            grad = buf
    param.add_(grad, alpha=-lr)
```

First of all, dampening isn't used in adamw so it's not really an exponential moving average of the same form. the default implementation is with dampening = 0 and this then means most recent update is way way larger than in momentum part of AdamW. I guess it's equivalent up to scaling But even ignoring this, the thing I'm more confused by is how the weight decay mechanism is added... So basically, let $\beta =$ weight decay, $\gamma =$ dampening, η_t, λ learning rate and wd terms and w_t, g_t the weights and gradients at time t Then in the AdamW implementation, the weight decay is applied directly to the parameters and then the update:

$$w_t = (1 - \eta_t \lambda) w_{t-1} - \eta_t \frac{m_t}{\sqrt{v_t} + \varepsilon} \quad \text{and} \quad m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

... makes sense

however the wd update insists

$$g_t = g_t + \lambda w_t$$

Chapter 4

A Brief History of Optimization

Chapter 5

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

Chapter 6

Bias Correction is Bogus

The Adam Optimizer was introduced in 2017. Since then, it has been the de facto stochastic optimizer choice for nearly all SOTA deep learning training (*along the way, weight decay decoupling was added to improve the optimizer).
The Adam optimizer itself is merely a synthesis of two already well-established optimization ideas - momentum and RMSProp. Adam reaps the benefits of both methods
The AdamW optimizer can be represented as follows:(need to replace this with the algorithm itself for sure)

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

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.
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 of "Necessity" of 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.
In any case, the influence of bias correction to the current EMA statistics is not appreciable after a sufficient number of steps are taken. Or so this may seem:

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.
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

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

Thus the first optimizer step is given by:

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

For the ZI and BC (and no ZI and no BC) it seems like basically a random first step. Next we have

$$m_2 = \beta_1 m_1 + (1-\beta_1)g_2 = \begin{cases} \beta_1(1-\beta_1)g_1 + (1-\beta_1)g_2, & \text{if ZI} \\ \beta_1 g_1 + (1-\beta_1)g_2 & \text{if no ZI} \end{cases}$$

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 ResNet50 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 https://github.com/Niccolo-Ajroldi/plainLM/tree/main of nanoGPT ([2])

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

Table 6.1: *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

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.

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

Some scratchings about learning rate perspective on AdamW

Discussed how bias correction looks a bit like lr scheduling

Chapter 7

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 which involves The general principle of stochastic optimizers in the age of deep learning is to extract local information about the curvature of the loss landscape without needing to explicitely compute the full second derivative.

We denote an arbitrary deep neural network as $f : \mathscr{X} \times \Theta \to \mathscr{Y}$ with $\Theta \subset \mathbb{R}^p$. 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 batch data and weights θ_t. This presentation views the parameters of a single vector.
The truth is a little bit more complicated. Deep neural networks consist of several layers

7.1 Understanding Muon

The idea for the Muon Optimizer originated from the

7.2 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.

7.2.1 Linear Regression

We consider the case of linear regression with a multi-dimensional output. This 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 then 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 will denote by G_t is given by

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

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

Bibliography

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

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