Thesis Notepad of Ideas and Progress etc

January 20, 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.

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

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

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