Understanding Adam and AdamW through Proximal Updates, Scale-Freeness, and Relaxed Smoothness

Francesco Orabona

KAUST

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Optimization in Deep Learning

- Many people in the deep learning community believe that optimization is not important
- This is clear looking at Google DeepMind: Not a single optimization team!

Optimization in Deep Learning

- Many people in the deep learning community believe that optimization is not important
- This is clear looking at Google DeepMind: Not a single optimization team!

- Yet, optimization is a critical component

Optimization in Deep Learning

- Many people in the deep learning community believe that optimization is not important
- This is clear looking at Google DeepMind: Not a single optimization team!

- Yet, optimization is a critical component
- But often optimization theory is too far from the reality of machine learning

Optimization in Deep Learning

- Many people in the deep learning community believe that optimization is not important
- This is clear looking at Google DeepMind: Not a single optimization team!

- Yet, optimization is a critical component
- But often optimization theory is too far from the reality of machine learning
- This talk is an attempt to bridge theory and practice and to show some interesting aspects of algorithms used in deep learning

A Motivating Example

- Scaling laws (Kaplan et al., 2020) “As more compute becomes available, we can choose how much to allocate towards training larger models, using larger batches, and training for more steps. [...] For optimally compute-efficient training, most of the increase should go towards increased model size.”

A Motivating Example

- Scaling laws (Kaplan et al., 2020) "As more compute becomes available, we can choose how much to allocate towards training larger models, using larger batches, and training for more steps. [...] For optimally compute-efficient training, most of the increase should go towards increased model size."
- Yet, they were wrong...

A Motivating Example

- Scaling laws (Kaplan et al., 2020) “As more compute becomes available, we can choose how much to allocate towards training larger models, using larger batches, and training for more steps. [...] For optimally compute-efficient training, most of the increase should go towards increased model size.”
- Yet, they were wrong...
- **...because they did not tune properly the learning rate!!** “First, the authors use a fixed number of training tokens and learning rate schedule for all models [...] result[ing] in underestimating the effectiveness of training models on less data than 130B tokens, and eventually contributes to the conclusion that model size should increase faster than training data size as compute budget increases.” (Hoffmann et al., 2022)

Outline

1. Why Studying Adam and AdamW?
2. Understanding Adam with Scale-Freeness
3. Understanding AdamW with Proximal Updates
4. Understanding Adam with Relaxed Smoothness

Why Studying Adam and AdamW?

- Adam and AdamW are the most used algorithms in deep learning
- Proof #1: Adam has 177150 citations, AdamW 16395 citations
- Proof #2: Most used ones even to train large language models

Model	Batch Size (#tokens)	Learning Rate	Warmup	Decay Method	Optimizer	Precision Type	Weight Decay	Grad Clip	Dropout
GPT3 (175B)	32K→3.2M	6×10^{-5}	yes	cosine decay to 10%	Adam	FP16	0.1	1.0	-
PanGu-α (200B)	-	2×10^{-5}	-	-	Adam	-	0.1	-	-
OPT (175B)	2M	1.2×10^{-4}	yes	manual decay	AdamW	FP16	0.1	-	0.1
PaLM (540B)	1M→4M	1×10^{-2}	no	inverse square root	Adafactor	BF16	lr^2	1.0	0.1
BLOOM (176B)	4M	6×10^{-5}	yes	cosine decay to 10%	Adam	BF16	0.1	1.0	0.0
MT-NLG (530B)	64 K→3.75M	5×10^{-5}	yes	cosine decay to 10%	Adam	BF16	0.1	1.0	-
Gopher (280B)	3M→6M	4×10^{-5}	yes	cosine decay to 10%	Adam	BF16	-	1.0	-
Chinchilla (70B)	1.5M→3M	1×10^{-4}	yes	cosine decay to 10%	AdamW	BF16	-	-	-
Galactica (120B)	2M	7×10^{-6}	yes	linear decay to 10%	AdamW	-	0.1	1.0	0.1
LaMDA (137B)	256K	-	-	-	-	BF16	-	-	-
Jurassic-1 (178B)	32 K→3.2M	6×10^{-5}	yes	-	-	-	-	-	-
LLaMA (65B)	4M	1.5×10^{-4}	yes	cosine decay to 10%	AdamW	-	0.1	1.0	-
GLM (130B)	0.4M→8.25M	8×10^{-5}	yes	cosine decay to 10%	AdamW	FP16	0.1	1.0	0.1
T5 (11B)	64K	1×10^{-2}	no	inverse square root	AdaFactor	-	-	-	0.1
ERNIE 3.0 Titan (260B)	-	1×10^{-4}	-	-	Adam	FP16	0.1	1.0	-
PanGu-Σ (1.085T)	0.5M	2×10^{-5}	yes	-	Adam	FP16	-	-	-

Zhao et al. “A Survey of Large Language Models”, ArXiv’23

Why Adam is the Best Algorithm?

- If we want to design better optimization algorithms, we have to understand why Adam and AdamW work so well

Why Adam is the Best Algorithm?

- If we want to design better optimization algorithms, we have to understand why Adam and AdamW work so well
- Caveat: This task might be ill-posed.
- My blogpost from December 2020: Adam might be the best algorithm, because we only keep using neural network architectures where Adam works!

PARAMETER-FREE LEARNING AND OPTIMIZATION ALGORITHMS

HOME INTRODUCTION TO ONLINE LEARNING ICML TUTORIAL ABOUT ME CONTACT

NEURAL NETWORKS (MAYBE) EVOLVED TO MAKE ADAM THE BEST OPTIMIZER

by bremen79

DEC 06
2020

Disclaimer: This post will be a little different than my usual ones. In fact, I won't prove anything and I will just briefly explain some of my conjectures around optimization in deep neural networks. Differently from my usual posts, it is totally possible that what I wrote is completely wrong 🙂

I have been working on online and stochastic optimization for a while, from a practical and empirical point of view. So, I was already in this field when Adam (Kingma and Ba, 2015) was proposed.

Outline

1 Why Studying Adam and AdamW?

2 Understanding Adam with Scale-Freeness

3 Understanding AdamW with Proximal Updates

4 Understanding Adam with Relaxed Smoothness

Adam Is Scale-Free

- Suppose to scale the first coordinate of your gradients by 10
- If you use gradient descent, this means that you update the first coordinate much more than the other ones
- To counteract the scaling, you should divide the learning rate of the first coordinate by 10

Adam Is Scale-Free

- Suppose to scale the first coordinate of your gradients by 10
- If you use gradient descent, this means that you update the first coordinate much more than the other ones
- To counteract the scaling, you should divide the learning rate of the first coordinate by 10
- But we can do better

Adam Is Scale-Free

- Suppose to scale the first coordinate of your gradients by 10
- If you use gradient descent, this means that you update the first coordinate much more than the other ones
- To counteract the scaling, you should divide the learning rate of the first coordinate by 10
- But we can do better

- Some optimization algorithms are adaptive to the scale of the features: scale-free (Orabona&Pál, 2015, 2018)
- *The update of these algorithms is completely independent from any multiplicative scaling of each coordinate of the gradients*

Scale-Freeness Through "Adaptive Learning Rates"

- AdaGrad (Duchi et al., 2010; McMahan&Streeter, 2010) introduced the idea of having per-coordinate learning rates depending on past stochastic gradients $\boldsymbol{g}_t$
- Learning rate at iteration t on coordinate i: $\frac{1}{\epsilon+\sqrt{\sum_{j=1}^{t} g_{j,i}^2}}$
- It is easy to see that if a coordinate is multiplied by a scalar, the learning rate is divided by the same scalar (if $\epsilon \approx 0$)

- Adam has the same behaviour:

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$$
$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$$
$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta_t \boldsymbol{m}_t/(\sqrt{\boldsymbol{v}_t}+\epsilon)$$

Scale-free Algorithms Have an Implicit Preconditioner

Scale-free Algorithms Have an Implicit Preconditioner

Theorem

Let f be a twice continuously differentiable function and $\boldsymbol{x}^*$ such that $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$. Then, let $\tilde{f}_\Lambda$ be the family of functions such that $\nabla \tilde{f}_\Lambda(\boldsymbol{x}^*) = \mathbf{0}$, and $\nabla^2 \tilde{f}_\Lambda(\boldsymbol{x}) = \Lambda \nabla^2 f(\boldsymbol{x})$, where $\Lambda = diag(\lambda_1, \dots, \lambda_d) \succeq 0$.
Then, running any scale-free optimization algorithm on f and $\tilde{f}_\Lambda$ will result exactly in the same iterates.

Corollary: Any dependency on the condition number of the scale-free algorithm will be reduced to the smallest condition number among all the functions $\tilde{f}_\Lambda$.

(Zhuang et al., TMLR'22)

Example with Quadratics

Corollary

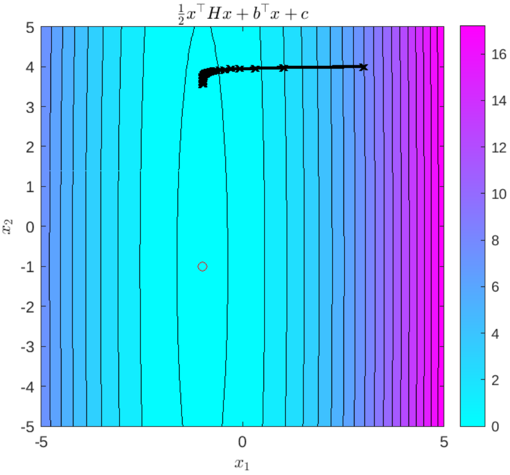
For quadratic problems with diagonal and positive definite Hessian, any scale-free algorithm will not differentiate between minimizing

- $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top H\boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{x} + c$
- $\tilde{f}(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{x} + (H^{-1}\boldsymbol{b})^\top \boldsymbol{x} + c.$

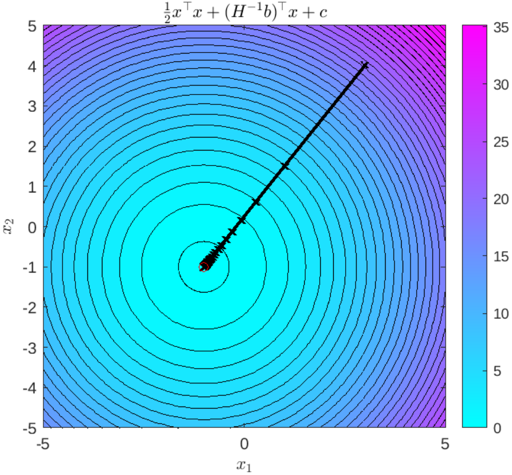
As the condition number of $\tilde{f}$ is 1, the convergence of a scale-free algorithm will not be affected by the condition number of f at all.

(Zhuang et al., TMLR'22)

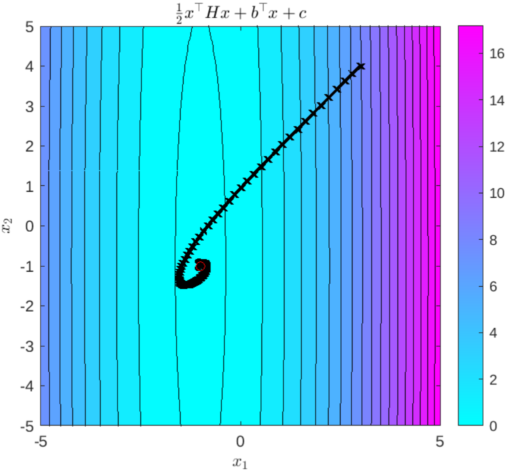
GD on a Quadratic



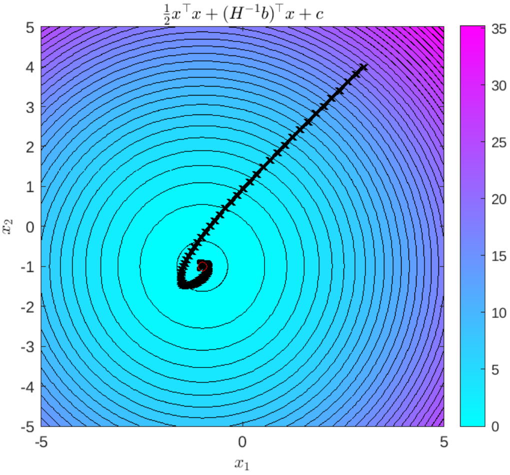
GD+Preconditioning on a Quadratic



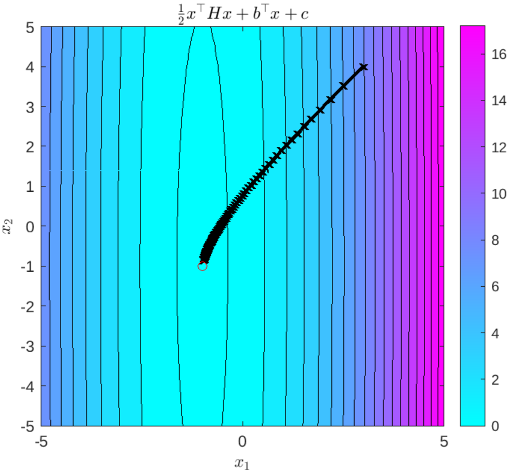
Adam on a Quadratic



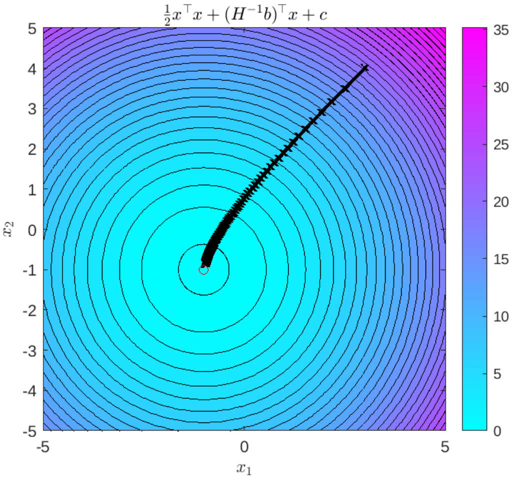
Adam+Preconditioning on a Quadratic



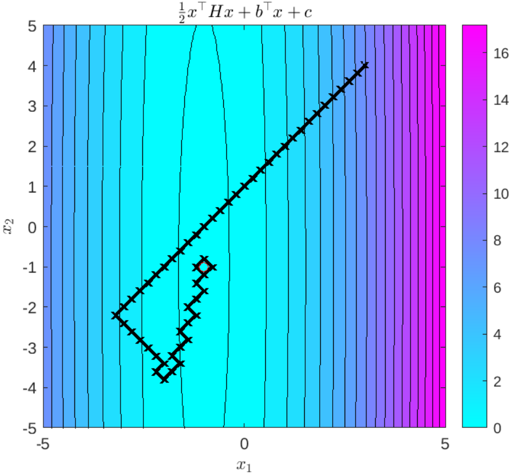
AdaGrad on a Quadratic



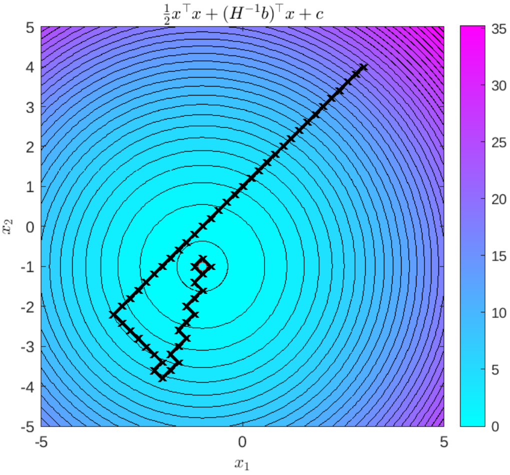
AdaGrad+Preconditioning on a Quadratic



Lion on a Quadratic



Lion+Preconditioning on a Quadratic



Take Home Messages I

- Adam has a scale-free update
- Scale-free updates have an “implicit preconditioner”
- This preconditioning effect does not depend on the presence of a “second-order term”, in fact it is present in Lion too

Outline

1 Why Studying Adam and AdamW?

2 Understanding Adam with Scale-Freeness

3 Understanding AdamW with Proximal Updates

4 Understanding Adam with Relaxed Smoothness

Squared L2 Regularization and Adam

- A commonly used regularizer is the squared L2 norm: $\text{Obj}(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|_2^2 + \text{TrainingLoss}(\boldsymbol{w})$

Squared L2 Regularization and Adam

- A commonly used regularizer is the squared L2 norm: $\text{Obj}(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|_2^2 + \text{TrainingLoss}(\boldsymbol{w})$
- Stochastic gradient calculated of loss plus regularizer is

$$\lambda \boldsymbol{w}_t + \underbrace{\frac{1}{m} \sum_{i=1}^{m} \nabla \ell(f_{\boldsymbol{w}_t}(\boldsymbol{x}_i, y_i))}_{\nabla_t = \text{Stochastic Gradient}}$$

Squared L2 Regularization and Adam

- A commonly used regularizer is the squared L2 norm: $\text{Obj}(\boldsymbol{w}) = \lambda \|\boldsymbol{w}\|_2^2 + \text{TrainingLoss}(\boldsymbol{w})$
- Stochastic gradient calculated of loss plus regularizer is

$$\underbrace{\lambda \boldsymbol{w}_t + \frac{1}{m}\sum_{i=1}^{m} \nabla \ell(f_{\boldsymbol{w}_t}(\boldsymbol{x}_i, y_i))}_{\nabla_t = \text{Stochastic Gradient}}$$

- SGD: $\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \lambda\eta \boldsymbol{w}_t - \eta\nabla_t = \underbrace{(1-\eta\lambda)\boldsymbol{w}_t}_{\text{weight decay}} - \eta\nabla_t$

Squared L2 Regularization and Adam

- A commonly used regularizer is the squared L2 norm: $\text{Obj}(\boldsymbol{w}) = \lambda\|\boldsymbol{w}\|_2^2 + \text{TrainingLoss}(\boldsymbol{w})$
- Stochastic gradient calculated of loss plus regularizer is

$$\lambda \boldsymbol{w}_t + \underbrace{\frac{1}{m}\sum_{i=1}^{m} \nabla \ell(f_{\boldsymbol{w}_t}(\boldsymbol{x}_i, y_i))}_{\nabla_t = \text{Stochastic Gradient}}$$

- SGD: $\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \lambda\eta \boldsymbol{w}_t - \eta\nabla_t = \underbrace{(1 - \eta\lambda)\boldsymbol{w}_t}_{\text{weight decay}} - \eta\nabla_t$
- Learning rate η and λ are now *linked*: If $\lambda\eta > 1$, the sign of $\boldsymbol{w}_t$ flips and it might even grow instead of shrinking!

Adam vs AdamW

- Adam update

$$\begin{aligned} \boldsymbol{g}_t &= \lambda \boldsymbol{w}_{t-1} + \nabla_t \\ \boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2 \\ \boldsymbol{w}_t &= \boldsymbol{w}_{t-1} - \eta_t \boldsymbol{m}_t / (\sqrt{\boldsymbol{v}_t} + \epsilon) \end{aligned}$$

- A different heuristic: AdamW

$$\begin{aligned} \boldsymbol{g}_t &= \nabla_t \\ \boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2 \\ \boldsymbol{w}_t &= \boldsymbol{w}_{t-1} - \eta_t (\lambda \boldsymbol{w}_{t-1} + \boldsymbol{m}_t / (\sqrt{\boldsymbol{v}_t} + \epsilon)) \end{aligned}$$

- Motivation: “decouples” η and λ

(Loshchilov&Hutter, 2019)

Small Detour: Proximal Updates

- Where the gradient descent update comes from?

$$\boldsymbol{w}_t = \operatorname*{argmin}_{\boldsymbol{w}} \underbrace{f(\boldsymbol{w}_{t-1}) + \langle \nabla f(\boldsymbol{w}_{t-1}), \boldsymbol{w} - \boldsymbol{w}_{t-1} \rangle}_{\text{Taylor approximation around } \boldsymbol{w}_{t-1}} + \underbrace{\frac{1}{2\eta} \|\boldsymbol{w} - \boldsymbol{w}_{t-1}\|_2^2}_{\text{Stay close to } \boldsymbol{w}_{t-1}}$$

$$= \boldsymbol{w}_{t-1} - \eta \nabla f(\boldsymbol{w}_{t-1})$$

Small Detour: Proximal Updates

- Where the gradient descent update comes from?

$$\boldsymbol{w}_t = \operatorname*{argmin}_{\boldsymbol{w}} \underbrace{f(\boldsymbol{w}_{t-1}) + \langle \nabla f(\boldsymbol{w}_{t-1}), \boldsymbol{w} - \boldsymbol{w}_{t-1} \rangle}_{\text{Taylor approximation around } \boldsymbol{w}_{t-1}} + \underbrace{\frac{1}{2\eta} \|\boldsymbol{w} - \boldsymbol{w}_{t-1}\|_2^2}_{\text{Stay close to } \boldsymbol{w}_{t-1}}$$

$$= \boldsymbol{w}_{t-1} - \eta \nabla f(\boldsymbol{w}_{t-1})$$

- A better update through Proximal Updates

$$\boldsymbol{w}_t = \operatorname*{argmin}_{\boldsymbol{w}} \underbrace{f(\boldsymbol{w})}_{\text{Actual function}} + \underbrace{\frac{1}{2\eta} \|\boldsymbol{w} - \boldsymbol{w}_{t-1}\|_2^2}_{\text{Stay close to } \boldsymbol{w}_{t-1}}$$

- No closed form in most of the cases: as difficult as minimizing f!
- Yet, better theoretical and empirical performance

An Efficient Variant: Partial Linearization

- If the loss function is composed by two parts, for example regularizer + loss, we can linearize only one part
- For example, we can linearize only the loss

$$
\begin{aligned}
\boldsymbol{w}_t &= \operatorname*{argmin}_{\boldsymbol{w}} \ \underbrace{\frac{\lambda}{2}\|\boldsymbol{w}\|_2^2}_{\text{Full regularizer}} + \underbrace{f(\boldsymbol{w}_{t-1}) + \langle \nabla f(\boldsymbol{w}_{t-1}), \boldsymbol{w} - \boldsymbol{w}_t \rangle}_{\text{Taylor approximation } f \text{ around } \boldsymbol{w}_{t-1}} + \underbrace{\frac{1}{2\eta}\|\boldsymbol{w} - \boldsymbol{w}_{t-1}\|_2^2}_{\text{Stay close to } \boldsymbol{w}_{t-1}} \\
&= \frac{\boldsymbol{w}_{t-1} - \eta \nabla f(\boldsymbol{w}_{t-1})}{1 + \lambda\eta}
\end{aligned}
$$

- Now η and λ are independent!
  This update will never flip the sign nor grow $\boldsymbol{w}$, for any learning rate $\eta \geq 0$

AdamW is an Approximated Proximal Step!

- Adam updates with a normalized momentum instead of gradient

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} + \eta \frac{\boldsymbol{m}_t}{\epsilon + \sqrt{\boldsymbol{v}_t}}$$

where the $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ contains the gradient of the regularizer too

AdamW is an Approximated Proximal Step!

- Adam updates with a normalized momentum instead of gradient

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} + \eta \frac{\boldsymbol{m}_t}{\epsilon + \sqrt{\boldsymbol{v}_t}}$$

  where the $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ contains the gradient of the regularizer too

- The proximal version of the same update is

$$\boldsymbol{w}_t = \frac{\boldsymbol{w}_{t-1} + \eta \frac{\boldsymbol{m}_t}{\epsilon + \sqrt{\boldsymbol{v}_t}}}{1 + \lambda\eta} = \underbrace{(1 - \lambda\eta)\boldsymbol{w}_{t-1} + \eta \frac{\boldsymbol{m}_t}{\epsilon + \sqrt{\boldsymbol{v}_t}}}_{\text{AdamW update}} + O(\eta^2)$$

AdamW is an Approximated Proximal Step!

- Adam updates with a normalized momentum instead of gradient

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} + \eta \frac{\boldsymbol{m}_t}{\epsilon + \sqrt{\boldsymbol{v}_t}}$$

  where the $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ contains the gradient of the regularizer too
- The proximal version of the same update is

$$\boldsymbol{w}_t = \frac{\boldsymbol{w}_{t-1} + \eta \frac{\boldsymbol{m}_t}{\epsilon + \sqrt{\boldsymbol{v}_t}}}{1 + \lambda\eta} = \underbrace{(1 - \lambda\eta)\boldsymbol{w}_{t-1} + \eta \frac{\boldsymbol{m}_t}{\epsilon + \sqrt{\boldsymbol{v}_t}}}_{\text{AdamW update}} + O(\eta^2)$$

- Bonus effect: Both updates above are scale-free, while Adam with L2 regularizer is not

AdamW is an Approximated Proximal Step!

- Adam updates with a normalized momentum instead of gradient

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} + \eta \frac{\boldsymbol{m}_t}{\epsilon + \sqrt{\boldsymbol{v}_t}}$$

  where the $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ contains the gradient of the regularizer too
- The proximal version of the same update is

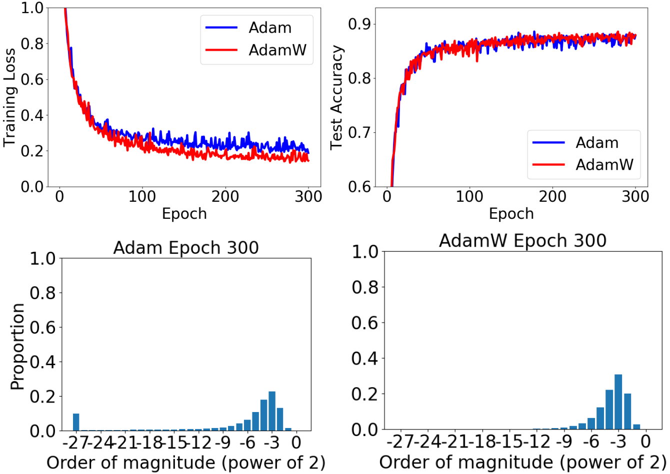
$$\boldsymbol{w}_t = \frac{\boldsymbol{w}_{t-1} + \eta \frac{\boldsymbol{m}_t}{\epsilon + \sqrt{\boldsymbol{v}_t}}}{1 + \lambda\eta} = \underbrace{(1 - \lambda\eta)\boldsymbol{w}_{t-1} + \eta \frac{\boldsymbol{m}_t}{\epsilon + \sqrt{\boldsymbol{v}_t}}}_{\text{AdamW update}} + O(\eta^2)$$

- Bonus effect: Both updates above are scale-free, while Adam with L2 regularizer is not
- We can now design the “AdamW version” of any other algorithm: just use the proximal view

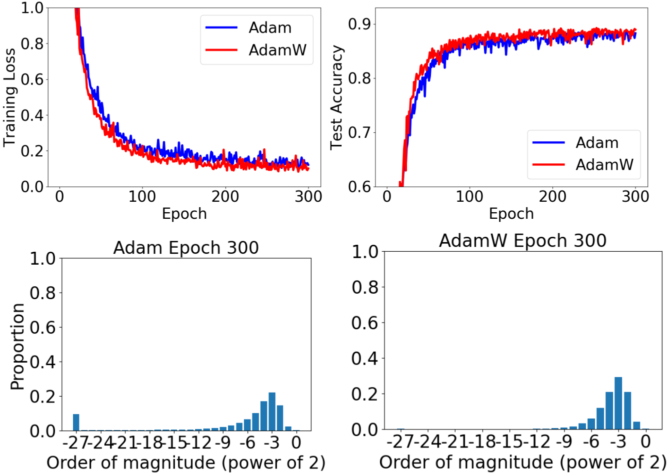
Scale-Freeness Correlates with Better Performance

- Deep learning people have developed a number of tricks to have all the weights roughly in the same ranges
- Batch normalization (BN) is the most used heuristic to accomplish it (Ioffe&Szegedy, ICML'15)
- BN is so effective that makes AdamW useless (Bjorck et al. AAAI'21)
- But what happens without BN?

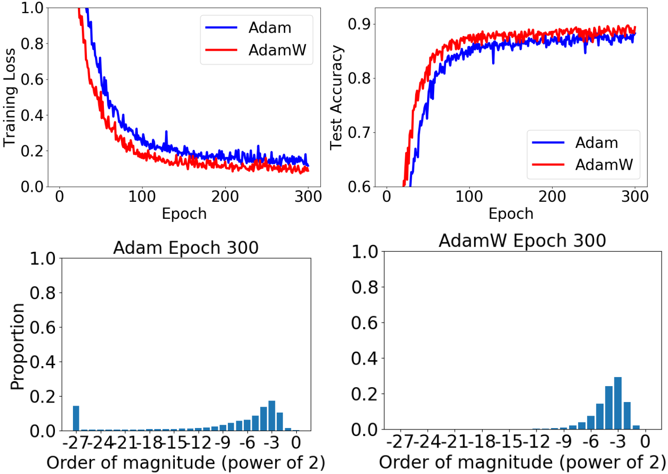
20 Layer Resnet on CIFAR10



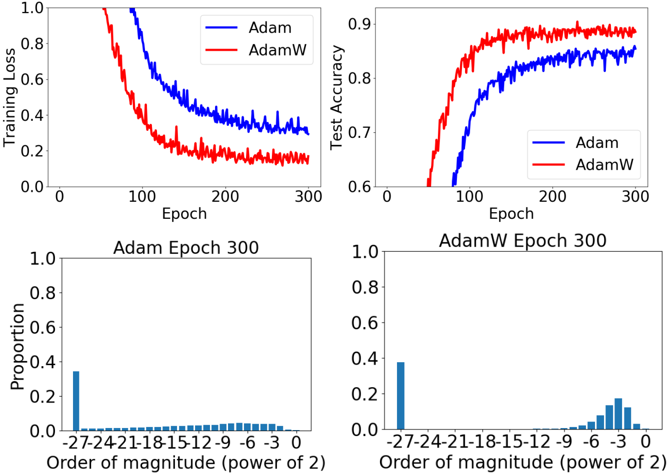
44 Layer Resnet on CIFAR10



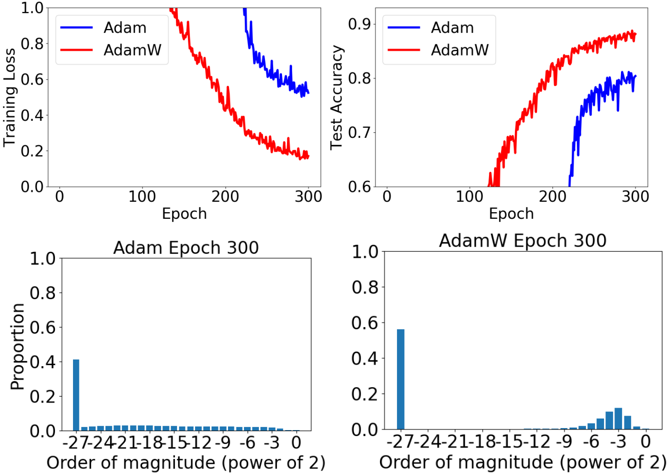
56 Layer Resnet on CIFAR10



110 Layer Resnet on CIFAR10



218 Layer Resnet on CIFAR10



Take Home Messages II

- Adam is not scale-free if we use a squared L2 regularizer
- Fix: AdamW, an approximate proximal update, is scale-free
- Any other algorithm can use a proximal update for the squared L2 regularizer!
- Scale-free updates correlates with better performance, in training and testing

Outline

1 Why Studying Adam and AdamW?

2 Understanding Adam with Scale-Freeness

3 Understanding AdamW with Proximal Updates

4 Understanding Adam with Relaxed Smoothness

How Do We Study Optimization Algorithm?

- No optimization algorithm can be better than all the others in all cases
- So we need to restrict to a family of functions, usually we consider *smooth* functions
- A differentiable function $F : \mathbb{R}^d \to \mathbb{R}$ is M-smooth if

$$\|\nabla F(\boldsymbol{x}) - \nabla F(\boldsymbol{y})\|_2 \leq M\|\boldsymbol{x} - \boldsymbol{y}\|_2$$

- In a smooth function:
  - The gradients go to zero approaching a minimum, even if the function is non-convex
  - The functions is upper bounded by a quadratic

(L_0, L_1)-Smoothness

However, smoothness is **not** a good characterization of the landscapes of deep neural networks training objectives [B. Zhang et al., ICLR'20]

(L_0, L_1)-Smoothness

However, smoothness is **not** a good characterization of the landscapes of deep neural networks training objectives [B. Zhang et al., ICLR'20]

- Relaxed smoothness [B. Zhang et al., ICLR'20]:

$$\|\nabla^2 F(\boldsymbol{x})\| \leq L_0 + L_1 \|\nabla F(\boldsymbol{x})\|, \ \forall \boldsymbol{x} \in \mathbb{R}^d$$

- No twice differentiable variant [J. Zhang et al., NeurIPS'20]:

$$\|\nabla F(\boldsymbol{x}) - \nabla F(\boldsymbol{y})\|_2 \leq (L_0 + L_1 \|\nabla F(\boldsymbol{x})\|_2) \|\boldsymbol{x} - \boldsymbol{y}\|_2,$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d : \|\boldsymbol{x} - \boldsymbol{y}\|_2 \leq \frac{1}{L_1}$

(L_0, L_1)-Smoothness

However, smoothness is **not** a good characterization of the landscapes of deep neural networks training objectives [B. Zhang et al., ICLR'20]

- Relaxed smoothness [B. Zhang et al., ICLR'20]:

$$\|\nabla^2 F(\boldsymbol{x})\| \leq L_0 + L_1 \|\nabla F(\boldsymbol{x})\|, \; \forall \boldsymbol{x} \in \mathbb{R}^d$$

- No twice differentiable variant [J. Zhang et al., NeurIPS'20]:

$$\|\nabla F(\boldsymbol{x}) - \nabla F(\boldsymbol{y})\|_2 \leq (L_0 + L_1 \|\nabla F(\boldsymbol{x})\|_2) \|\boldsymbol{x} - \boldsymbol{y}\|_2,$$

  for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d : \|\boldsymbol{x} - \boldsymbol{y}\|_2 \leq \frac{1}{L_1}$

- The curvature can increase far away from a local minimum

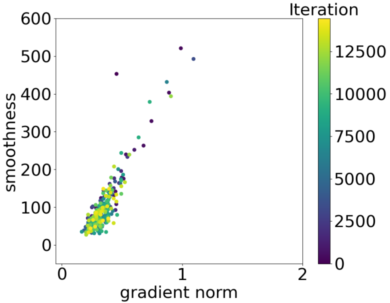
Why Relaxed Smoothness?

- Many interesting functions are non-smooth, but they are (L_0, L_1)-smooth
- Examples:
  - All univariate polynomials, like x^4
  - $\exp(x)$
- Relaxed smoothness means that the function could grow much faster than a quadratic, hence gradients can be very large

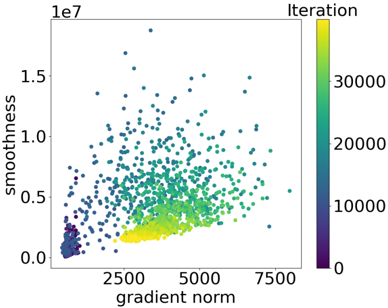
- More importantly, Zhang et al. [ICLR'20] empirically showed that this assumption holds for LSTMs

Transformers Satisfy Relaxed Smoothness Too

We show that this is true even on Transformers



(a) Wikitext-2



(b) WMT’16 de-en

(Crawshaw et al., NeurIPS’22)

SGD with Gradient Clipping under (L_0, L_1)-smoothness

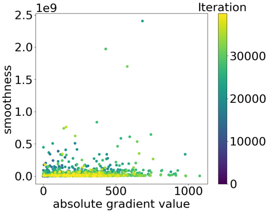
- Gradient clipping technique ensures SGD's convergence under (L_0, L_1)-smoothness [B. Zhang et al., ICLR'20]
- Gradient clipping is necessary because the relaxed smoothness can make the gradient exponentially big

SGD with Gradient Clipping under (L_0, L_1)-smoothness

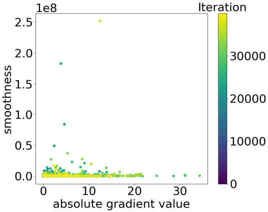
- Gradient clipping technique ensures SGD's convergence under (L_0, L_1)-smoothness [B. Zhang et al., ICLR'20]
- Gradient clipping is necessary because the relaxed smoothness can make the gradient exponentially big

- But...

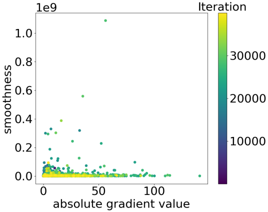
Relaxed Smoothness Changes a Lot across Layers!



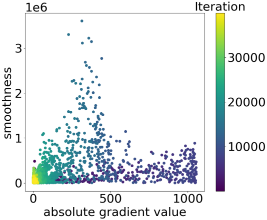
(a) Encoder First Layer



(b) Encoder Last Layer



(c) Decoder Second Layer



(d) Decoder Last Layer

WMT'16 de-en

(Crawshaw et al., NeurIPS'22)

A New Coordinate-wisely Relaxed Smooth Condition

Let $\boldsymbol{L}_0 := [L_{0,1}, \ldots, L_{0,d}]^T$ and $\boldsymbol{L}_1 := [L_{1,1}, \ldots, L_{1,d}]^T$. A differentiable function $F(\boldsymbol{x})$ is $(\boldsymbol{L}_0, \boldsymbol{L}_1)$-smooth coordinate-wisely, if for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ such that $\|\boldsymbol{x} - \boldsymbol{y}\|_2 \leq \frac{1}{\|\boldsymbol{L}_1\|_\infty}$, we have

$$\left|\frac{\partial F}{\partial x_j}(\boldsymbol{y}) - \frac{\partial F}{\partial x_j}(\boldsymbol{x})\right| \leq \left(\frac{L_{0,j}}{\sqrt{d}} + L_{1,j}\left|\frac{\partial F}{\partial x_j}(\boldsymbol{x})\right|\right)\|\boldsymbol{y} - \boldsymbol{x}\|_2, \ \forall j \in [d]$$

- Better model for reality
- You cannot hope to show an advantage of Adam-like updates over SGD without considering this coordinate-wise version!

(Crawshaw et al., NeurIPS'22)

A General Adam-like Algorithm

Algorithm Generalized SignSGD
(All operations on vectors are element-wise)

Inputs: $\boldsymbol{x}_1, \beta_1, \beta_2, \eta$
$\boldsymbol{m}_0 = 0, \boldsymbol{v}_0 = 0$
for $t = 1, \cdots, T$ **do**
  Compute $\boldsymbol{g}_t$, an unbiased estimate of $\nabla F(\boldsymbol{x}_t)$
  $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$
  $\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{m}_t^2$
  $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \frac{\boldsymbol{m}_t}{\sqrt{\boldsymbol{v}_t}}$
end for

- Difference with Adam: $\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
- Still Scale-free!
- If $\beta_2 = 0$ we get SignSGD with momentum

(Crawshaw et al., NeurIPS'22)

Theoretical Convergence Guarantee

Theorem

Assume F is $(\boldsymbol{L}_0, \boldsymbol{L}_1)$-coordinate-wise smooth and the noise on the stochastic gradient coordinate j is bounded by σ_j w.p. 1.
Then, there exist settings for η, β_1, β_2 and T large enough such that, Generalized SignSGD guarantees with high probability that

$$\min_{t\in[T]} \|\nabla F(\boldsymbol{x}_t)\|_1 = \tilde{\mathcal{O}}\left(\underbrace{\frac{\|\boldsymbol{L_0}\|_1^{\frac{1}{4}}\Delta^{\frac{1}{4}}\|\boldsymbol{\sigma}\|_1^{\frac{1}{2}}}{T^{\frac{1}{4}}}}_{\textit{Noise}} + \underbrace{\frac{\sqrt{\|\boldsymbol{L_0}\|_1\Delta}}{\sqrt{T}}}_{\textit{Smoothness}}\right)$$

$$+\tilde{\mathcal{O}}\left(\underbrace{(\|\boldsymbol{M}\|_1 + \|\boldsymbol{\sigma}\|_1)\exp\left(-\frac{\|\boldsymbol{L_0}\|_1^{3/4}}{\|\boldsymbol{L_1}\|_\infty\|\boldsymbol{\sigma}\|_1^{1/2}\Delta^{1/4}}T^{1/4}\right)}_{\textit{Relaxed Smoothness + Unbounded Gradients}}\right),$$

where $M_j := \sup\left\{\left|\frac{\partial F}{\partial x_j}(\boldsymbol{x})\right| : F(\boldsymbol{x}) \le F(\boldsymbol{x}_1)\right\} < \infty$, $\Delta := F(\boldsymbol{x}_1) - F^*$.

(Crawshaw et al., NeurIPS'22)

Lower Bound of GD

Theorem

Fix $\epsilon > 0, L_0 > 0, L_1 > 0, M \geq \max(\frac{L_0}{L_1}, \epsilon)$, and $x_0 \in \mathbb{R}$. Pick any constant learning rate η for GD, with the knowledge of the above constants.
Then, there exists a 1-d (L_0, L_1)-smooth function F, bounded from below by F^*, such that $\sup\{|F'(x)| : F(x) \leq F(x_0)\} \leq M$ on which the number of iterations T of GD with learning rate η to guarantee $|F'(x_T)| < \epsilon$ is at least

$$\frac{{\color{red}M} L_1 (F(x_0) - F^* - \frac{15\epsilon^2}{16 L_0})}{2\epsilon^2 \left(\ln \frac{M L_1}{L_0} + 1\right)}$$

While, generalized SignSGD rate is $\tilde{O}(\frac{L_0(F(x_0) - F^*)}{\epsilon^2})$

(Lower bound in Zhang et al. (NeurIPS'20) has an error, we fixed it)

Experiments Setup

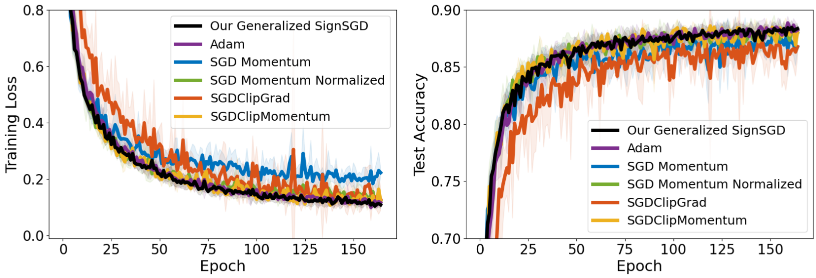
Competitors:

1. Adam
2. SGD Momentum: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \boldsymbol{m}_t$
3. SGD Momentum Normalized: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \frac{\boldsymbol{m}_t}{\|\boldsymbol{m}_t\|_2}$
4. SGDClipGrad: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \min\left(\eta, \frac{\gamma}{\|\boldsymbol{g}_t\|_2}\right) \boldsymbol{g}_t$
5. SGDClipMomentum: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \min\left(\eta, \frac{\gamma}{\|\boldsymbol{m}_t\|_2}\right) \boldsymbol{m}_t$

(The momentum term is $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$)

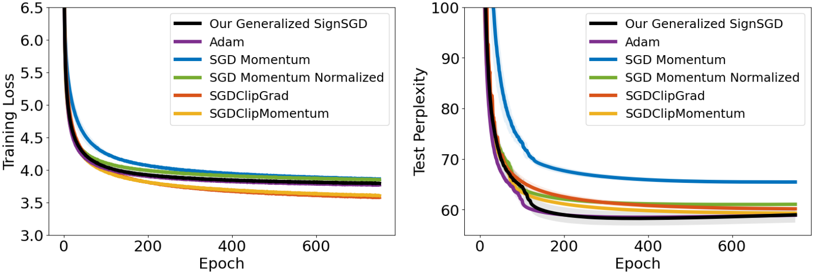
(Kingma & Ba, ICLR'15; Zhang et al., NeurIPS'20; Zhang et al., NeurIPS'21)

Resnet on CIFAR10



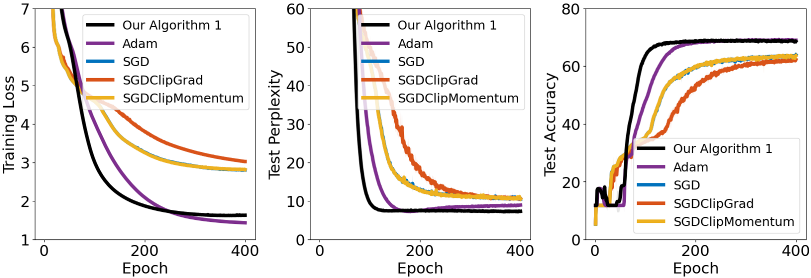
- Train the 20-layer Residual Network model to do image classification on the CIFAR-10 dataset
  - Mini-batch size is 128
  - No learning rate schedule
  - Training+testing with best hyperparameters repeated 5 times with different random seeds

LSTM on Penn Treebank



- Train a 3-layer AWD-LSTM to do language modeling (word level) on the Penn Treebank dataset
  - Mini-batch size is 40
  - No learning rate schedule
  - Training+testing with best hyperparameters repeated 5 times with different random seeds

Transformer on Translation Task



- Train a 6-layer Transformer on WMT'16 German-English Translation Task
  - Mini-batch size is 256
  - Learning rate warm-up and decay
  - Training+testing with best hyperparameters repeated 5 times with different random seeds

Take Home Messages III

- Relaxed smoothness is a closer assumption to the real world
- It allows to prove that a (minor) variant of Adam is provable better than SGD

Summary

- To design the next generation of optimization algorithms, we should understand why the current algorithms work
- Assumptions are also crucial for our theoretical analyses

- Some (unusual?) perspectives: scale-freeness, proximal updates, and relaxed smoothness

Thanks for your attention

Thanks to my collaborators: Michael Crawshaw, Ashok Cutkosky, Mingrui Liu, Wei Zhang, Zhenxun Zhuang

`http://francesco.orabona.com`