Optimization

- □□ Muon Optimizer: Un Enfoque Basado en Newton-Schulz
- The Road Less Scheduled
- □□ APOLLO: SGD-Like Memory, AdamW-Level Performance
- □□Physics-informed Knowledge Transfer for Underwater Monocular Depth Estimation

□□ Muon Optimizer: Un Enfoque Basado en Newton-Schulz

□□ Integrantes:

- □□ Andrea Parra
- □□ Jorge Andrey

□□ Material de apoyo:

- □□ **Diapositivas:** Ver presentaciones
- □□ **Paper:** Modular Duality in Deep Learning
- □□ **Código externo:** TBA
- □□ **Blog muon:** Muon: An optimizer for hidden layers in neural networks

El optimizador **Muon** es una variante del método de descenso de gradiente con momentum (SGD+Momentum) que introduce una corrección basada en la iteración de **Newton-Schulz** para mejorar la actualización de parámetros. Esta técnica permite estabilizar los gradientes y normalizar la actualización sin necesidad de cálculos costosos como la descomposición en valores singulares (SVD).

En esta sección exploraremos la motivación detrás de Muon, su estructura matemática y cómo Newton-Schulz contribuye a su eficacia.

□□ Antecedentes

Este método se basa en el método Newton-Schulz, utilizado para aproximar de la operación , donde es una matriz ortogonal. Sin embargo, los autores del paper proponen una modificación para obtener , lo que permite la normalización de gradientes en redes neuronales profundas.

Muon busca **aproximar la función de signo de la matriz de gradientes**, lo cual puede interpretarse como "ajustar los valores singulares a 1", asegurando que la actualización de

parámetros mantenga una estructura ortogonal. Este proceso se conoce como "symmetric orthogonalization" y se diferencia del Gram-Schmidt porque no favorece una fila o columna específica.

Los autores eligieron Newton-Schulz por su capacidad para ejecutarse de manera estable en bfloat16, a diferencia de la SVD y otras iteraciones de Newton más costosas o inestables en GPUs modernas.

□□ Objetivos

□ Entender la importancia de métodos de optimización como Muon.
□ (Intentar) aprender la matemática detrás de la optimización.
□ Abrir la discusión sobre aplicaciones futuras.

□□ Motivación y Contexto

Los métodos de optimización convencionales como SGD y Adam pueden sufrir de **desaparición o explosión del gradiente**, especialmente en redes neuronales profundas. Esto se debe a que la propagación del gradiente puede amplificar valores en ciertas direcciones, afectando la convergencia y estabilidad del entrenamiento.

Los autores de Muon encontraron que en modelos **transformer-based**, las actualizaciones de SGD-momentum y Adam tienen un número de condición muy alto. Es decir, las actualizaciones de pesos están **dominadas por pocas direcciones**, lo que limita la capacidad de aprendizaje. La ortogonalización mediante Newton-Schulz **aumenta la escala de direcciones poco representadas**, ayudando a mejorar la optimización.

⚙ Método Newton-Schulz en Muon

La iteración de Newton-Schulz es un método iterativo que aproxima la inversa de una matriz sin requerir una factorización directa. En el contexto del optimizador Muon, se utiliza para normalizar el gradiente acumulado B_T antes de actualizar los parámetros.

Iteración Newton-Schulz Rectangular

Dado un gradiente acumulado (B_t), se inicia con:

$$ X_0 = \frac{B_t}{||B_t||_F} $$

Luego, se aplica la siguiente iteración para aproximar UV^T de la SVD de B_t:

X_t

Este método garantiza que los valores singulares se ajusten de manera controlada y que la iteración converja de manera estable, incluso en matrices de bajo rango.

Luego, los autores cambian la fórmula para utilizar una expansión de un polinomio, de la siguiente manera.

X_t_2

Tomando solo hasta el término cuadrático, y remplanzando X por su SVD, tenemos (en la imagen X es G)

G'

Luego, si hacemos que la función interna tienda a la función signo, tenemos.

drawing
4

Si la función signo tiende a 1, tenemos una aproximación a UV^T, que es la matriz ortogonal del gradiente.

Los autores también exploraron el ajuste de coeficientes de Newton-Schulz para acelerar la convergencia, logrando reducir la cantidad de iteraciones necesarias a solo 5 en sus experimentos.

□□ Diferencias entre Muon y SGD+Momentum

La principal diferencia de Muon con SGD+Momentum es la inclusión del término **(O_t)**, obtenido mediante Newton-Schulz:

SGD+Momentum (Convencional)

Require: Learning rate η, momentum μ

1: Inicializar parámetros θ_0

2: Inicializar velocidad $v_0 \leftarrow 0$

3: for t = 1, ... do

4: Calcular gradiente $G_t \leftarrow \nabla\theta \mathcal{L}_t(\theta_{t-1})$

5: Actualizar velocidad $v_t \leftarrow \mu\, v_{t-1} + G_t$

6: Actualizar parámetros $\theta_t \leftarrow \theta_{t-1} - \eta\, v_t$

7: end for

8: return θ_t

Muon Optimizer

Require: Learning rate η, momentum μ

1: Inicializar $B_0 \leftarrow 0$

2: for t = 1, ... do

3: Calcular gradiente $G_t \leftarrow \nabla\theta \mathcal{L}_t(\theta_{t-1})$

4: $B_t \leftarrow \mu B_{t-1} + G_t$

5: $O_t \leftarrow \text{NewtonSchulz5}(B_t)$

6: Actualizar parámetros $\theta_t \leftarrow \theta_{t-1} - \eta O_t$

7: end for

8: return θ_t

- En **SGD+Momentum**, el gradiente acumulado se usa directamente para actualizar los parámetros.
- En **Muon**, el gradiente acumulado B_t es corregido mediante Newton-Schulz para obtener O_t, asegurando una actualización bien condicionada.

Esto ayuda a evitar direcciones de gradiente mal condicionadas y estabiliza la convergencia.

□□ Aplicaciones y Beneficios

El uso de Muon y Newton-Schulz tiene aplicaciones en diversas áreas:

 Redes neuronales profundas: Mejora la estabilidad en entrenamientos largos.
 Redes recurrentes (RNNs, Transformers): Evita problemas de explosión/desaparición del gradiente.
 Generative Adversarial Networks (GANs): Regulariza el entrenamiento para mejorar la calidad de las muestras generadas.
 Optimización de alto rendimiento: Reduce la necesidad de ajustes manuales en el learning rate.

Ventajas clave de Muon:

✔ Evita el costo computacional de la SVD.
✔ Mantiene estabilidad en gradientes.
✔ Regulariza la actualización de parámetros.

Referencias

Webpage autores de Muon sobre Newton-Schulz

Explicación en twitter

Higham, N. J. Functions of Matrices. SIAM, 2008.

The Road Less Scheduled

□□ Integrantes:

- □□ Juan José Calderón.
- □□ Julián David León Quintero.

□□ Material de apoyo:

- □□ **Diapositivas:** Ya es pronto :)
- □□ **Paper:** The Road Less Scheduled
- □□ **Código externo:** Repositorio de código
- ▶ **Video-explicación:** hu-po
- □□ **Canción para el despecho después de tanta matemática: Albertico**

□□ Objetivos

□□ En esta sección se definen los objetivos de la sesión:

□ Comprender la brecha teoría-práctica en optimización, especialmente en métodos como Polyak-Ruppert y los schedules de tasa de aprendizaje.

□ Posibilidad de respuesta a la pregunta principal: ¿Existen métodos de promediado que igualen el rendimiento práctico de los schedules, sin perder las garantías teóricas?

□ Obtener la intuición del diseño del método Schedule-Free: las ecuaciones clave, el papel del promedio de línea, y el momentum generalizado.

□ Analizar las garantías teóricas de los teoremas usados y cómo resuelven limitaciones de los métodos previos a este.

□□ Resultados Esperados

□□ Al final de la sesión, los asistentes podrán:

□□ Entender por qué Schedule-Free elimina la necesidad de schedules y cómo aprovecha el promedio dinámico para convergencia robusta.

□□ Diferenciar entre Polyak-Ruppert, Primal Averaging y Linear Decay como casos especiales del Teorema 2: Online-to-Batch generalizado.

□□ Comprender el impacto del parámetro beta en la estabilidad y velocidad de convergencia.

□□ Referencias Clave

□□ Esta sección recopila enlaces a recursos relevantes sobre procesamiento de imágenes:

□□ □□ **Teoría**: Polyak (1990), Cesa-Bianchi et al. (2004).

□□ □□**Momentum**: Nesterov (2018), Sutskever et al. (2013).

□□ □□ **Implementación**: Repo Schedule-Free.

□□ □□ **Evaluación**: MLCommons AlgoPerf (2024)

□□ APOLLO: SGD-Like Memory, AdamW-Level Performance

□□ Integrantes:

- □□ Henry Mantilla
- □□ Sebastián Solano

□□ Material de apoyo:

- □□ **Diapositivas:** Ver presentaciones
- □□ **Paper:** APOLLO: SGD-like Memory, AdamW-level Performance
- □□ **Código externo:** Repositorio de código

□□ Objetivos

□□ En esta sección se definen los objetivos de la sesión:

□ Destacar cómo la alta demanda de memoria en optimizadores como AdamW limita el entrenamiento de grandes modelos.

□ Explicar brevemente que APOLLO utiliza actualizaciones estructuradas (a nivel de canal o tensor) y proyecciones aleatorias de bajo rango para reducir el consumo de memoria.

□ Mostrar que APOLLO (y su variante Mini) logran resultados comparables o superiores a AdamW, pero con un coste de memoria similar al del SGD.

□□ Resultados Esperados

□□ Esta sección describe de manera general lo que se espera obtener al final de la sesión:

□□ Aprender a analizar las métricas de entrenamiento como perplexity para evaluar la eficiencia y el ahorro de memoria obtenido con APOLLO.

□□ Contrastar las diferencias clave entre APOLLO, AdamW y SGD, identificando las ventajas y limitaciones de cada uno.

□□ Los participantes podrán contrastar la complejidad y los costos de realizar una SVD exacta frente a la utilización de métodos de proyección de bajo rango, y argumentar por qué una aproximación estructurada es suficiente para entrenar LLMs.

□□ Referencias

□□ Esta sección recopila enlaces a recursos relevantes sobre procesamiento de imágenes:

□□ □□ Teoría

□□ □□ Optimizing LLMs for Speed and Memory

Physics-informed Knowledge Transfer for Underwater Monocular Depth Estimation

Integrantes:

- Sebastian Solano
- Brayan Quintero

Material de apoyo:

- **Diapositivas:** Ver presentaciones
- **Paper:** Ver artículos académicos
- **Código externo:** Repositorio de código

Objetivos

En esta sección se definen los objetivos de la sesión:
¿Por qué es importante este tema?
¿Qué se espera lograr durante la sesión?

Resultados Esperados

Esta sección describe de manera general lo que se espera obtener al final de la sesión:

Mayor comprensión del tema tratado.
Identificación de conceptos clave.

□□ Recopilación de información relevante para futuras implementaciones.

⚙ Metodología

□□ Aquí se explicarán todos los temas tratados en la sesión con mayor detalle. Esta sección se completará después de la sesión e incluirá:

□□ Explicaciones detalladas del proceso.
□□ Análisis de los conceptos presentados.
□□ Ejemplos prácticos y fragmentos de código.

□□ Ejemplo de código en Python:

```
import cv2
import matplotlib.pyplot as plt


imagen = cv2.imread(".images/ejemplo.png")
plt.imshow(cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB))
plt.show()
```

□□ Uso de imágenes
⚠ Solo utilizar imágenes disponibles en internet debido a las limitaciones de almacenamiento.

□□ Ejemplo de imagen adjunta:
Ejemplo de imagen

□□ También puedes ajustar el tamaño y alineación de las imágenes:
drawing

drawing

□□ Ejemplo de tabla:

□□ A	□□ B	□□ C
✔ Uno	Texto de prueba	□□

□□ Referencias

□□ Esta sección recopila enlaces a recursos relevantes sobre procesamiento de imágenes:

□□ □□ Documentación de OpenCV

□□ □□ Guía de NumPy

□□ □□ Artículo sobre procesamiento de imágenes