Implementación de Técnicas de Descenso de Gradiente Estocástico y Variantes: Una Perspectiva desde el Análisis Numérico

Una Comparación Experimental y Teórica

Brandon Trigueros

Curso de Análisis Numérico
Facultad de Ingeniería
Universidad de Costa Rica

25 de junio de 2025

Índice

¿Por qué importa el Descenso de Gradiente?

¡Está en TODAS partes!

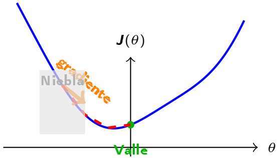
- **Google**: PageRank optimiza rankings de páginas web
- **Netflix**: Sistemas de recomendación personalizados
- **Tesla**: Conducción autónoma y visión computacional
- **ChatGPT**: Entrenamiento de modelos de lenguaje masivos
- **Medicina**: Diagnóstico por imágenes médicas (rayos X, resonancias)

El Desafío

- **Problema**: Optimizar funciones con millones/billones de parámetros
- **Datasets**: Terabytes de información (Twitter, YouTube, Amazon)
- **Tiempo**: Entrenar GPT-3 costó $4.6 millones en cómputo

Pregunta clave: ¿Cómo hacer que estos algoritmos sean MÁS RÁPIDOS y EFICIENTES?

La Analogía del Montañista



Montañista perdido en la niebla:

- Solo ve el **terreno local**
- Quiere llegar al **valle más bajo**
- Estrategia: seguir la **pendiente más empinada**

En Machine Learning:

- **Montaña** = función de error
- **Posición** = parámetros
- **Pendiente** = gradiente
- **Valle** = mejor solución

Objetivos del Trabajo

Objetivo Principal

Comparar experimentalmente cuatro técnicas de optimización:

- SGD básico
- SGD con Momentum
- RMSProp
- Adam

Metodología

- Implementación en Python desde cero
- Experimento con regresión logística en dataset Iris
- Análisis de curvas de convergencia

Conexión con Análisis Numérico

- Comparación con métodos de segundo orden
- Análisis de convergencia y trade-offs computacionales

Descenso de Gradiente Clásico

Fórmula Básica

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)$$

Ventajas:

- Convergencia estable
- Garantiza llegar al mínimo (funciones convexas)

Desventajas:

- Muy lento con datasets grandes
- Calcula gradiente completo en cada paso

Donde: η = tasa de aprendizaje, $J(\boldsymbol{\theta})$ = función de costo

SGD: Descenso de Gradiente Estocástico

Idea Principal

Usar solo **una muestra** (o pequeño mini-lote) por iteración:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}_t; \mathbf{x}_{i(t)}, y_{i(t)})$$

Trade-off Fundamental

- + Mucho más rápido computacionalmente
- + Permite manejar datasets enormes
- - Introduce ruido en las actualizaciones
- - Trayectoria más errática

SGD con Momentum

Analogía Física

Como una bola rodando que **acumula velocidad** y mantiene inercia

Fórmulas

$$v_t = \gamma v_{t-1} + \eta \nabla J(\boldsymbol{\theta}_t) \tag{1}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - v_t \tag{2}$$

Beneficios:

- Acelera en direcciones consistentes
- Amortigua oscilaciones

Riesgo:

- Puede "pasar de largo.[el] mínimo
- Requiere ajuste cuidadoso de η

Típicamente: $\gamma = 0{,}9$ (retiene 90 % de la velocidad previa)

RMSProp

Problema que Resuelve

Diferentes parámetros pueden necesitar diferentes tasas de aprendizaje

Fórmulas

$$E[g_j^2]_t = \rho E[g_j^2]_{t-1} + (1-\rho) g_{j,t}^2 \tag{3}$$

$$\theta_{j,t+1} = \theta_{j,t} - \frac{\eta}{\sqrt{E[g_j^2]_t + \varepsilon}} g_{j,t} \tag{4}$$

Intuición

- Si un parámetro tiene gradientes grandes $\Rightarrow$ paso más pequeño
- Si un parámetro tiene gradientes pequeños $\Rightarrow$ paso más grande
- **Adaptación automática** por coordenada

Típicamente: $\rho = 0{,}9$, $\varepsilon = 10^{-8}$

Adam: Lo Mejor de Dos Mundos

Combinación Inteligente

Adam = Momentum + RMSProp

Fórmulas (simplificadas)

$$m_{j,t} = \beta_1 m_{j,t-1} + (1-\beta_1) g_{j,t} \quad \text{(momentum)} \tag{5}$$

$$v_{j,t} = \beta_2 v_{j,t-1} + (1-\beta_2) g_{j,t}^2 \quad \text{(normalización)} \tag{6}$$

$$\theta_{j,t+1} = \theta_{j,t} - \frac{\eta}{\sqrt{\hat{v}_{j,t}} + \varepsilon} \hat{m}_{j,t} \tag{7}$$

¿Por qué es Popular?

- Funciona bien .ºut-of-the-box"
- Pocos hiperparámetros que ajustar
- Robusto en muchos problemas

Valores por defecto: $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$, $\eta = 0{,}001$

Diseño Experimental

Dataset Iris

- 150 muestras de flores Iris
- 4 características: longitud/ancho sépalos y pétalos
- **Clasificación binaria**: Versicolor vs. Virginia
- División: 80 % entrenamiento, 20 % prueba

Modelo: Regresión Logística

- Función sigmoide: $h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$
- Función de costo: Entropía cruzada binaria
- Gradiente analítico: $\nabla J = \frac{1}{N}\sum_i (h(\mathbf{x}_i) - y_i)\mathbf{x}_i$

Implementación en Python

SGD Básico

```
for epoch in range(epochs):
    for i in range(N):
        y_pred = sigmoid(np.dot(w, X[i]))
        grad = (y_pred - y[i]) * X[i]
        w = w - lr * grad
```

SGD con Momentum

```
v = np.zeros(d)  # velocidad inicial
for epoch in range(epochs):
    for i in range(N):
        grad = compute_gradient(w, X[i], y[i])
        v = gamma * v + lr * grad
        w = w - v
```

Configuración de Hiperparámetros

Después de experimentación, se eligieron:

Algoritmo	**Tasa de Aprendizaje**	**Parámetros Adicionales**
SGD	0.05	-
SGD + Momentum	0.03	$\gamma = 0{,}9$
RMSProp	0.05	$\rho = 0{,}9,\ \varepsilon = 10^{-8}$
Adam	0.05	$\beta_1 = 0{,}9,\ \beta_2 = 0{,}999$

Nota Importante

Momentum requirió menor tasa de aprendizaje para evitar inestabilidad

Método de Newton-Raphson vs. Descenso de Gradiente

Newton-Raphson (Segundo Orden)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathrm{H}^{-1}(\boldsymbol{\theta}_t)\nabla J(\boldsymbol{\theta}_t)$$

donde H es la matriz Hessiana (segundas derivadas)

Descenso de Gradiente (Primer Orden)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\nabla J(\boldsymbol{\theta}_t)$$

Newton-Raphson:

- Convergencia cuadrática
- Pocas iteraciones
- $O(n^3)$ por iteración
- Hessiana puede no ser positiva definida

Descenso de Gradiente:

- Convergencia lineal
- Más iteraciones
- $O(n)$ por iteración
- Siempre estable

¿Por qué no usar siempre Newton-Raphson?

El Problema de Escalabilidad

Para un modelo con n parámetros:

- **Gradiente**: vector de tamaño $n \to O(n)$ memoria
- **Hessiana**: matriz de tamaño $n \times n \to O(n^2)$ memoria
- **Inversión**: $O(n^3)$ operaciones

Ejemplo Real: GPT-3

- **Parámetros**: 175 mil millones ($n = 1{,}75 \times 10^{11}$)
- **Hessiana**: $(1{,}75 \times 10^{11})^2 = 3 \times 10^{22}$ elementos
- **Memoria**: $\sim 10^{14}$ TB (¡imposible!)

Conclusión: Necesitamos métodos de primer orden escalables $\to$ Descenso de Gradiente

Métodos Quasi-Newton: El Punto Intermedio

Idea Principal

Aproximar la inversa de la Hessiana sin calcularla explícitamente

Métodos Quasi-Newton:

- BFGS, L-BFGS
- Aproximan H^{-1} iterativamente
- Convergencia superlineal
- $O(n^2)$ memoria

Métodos Adaptativos:

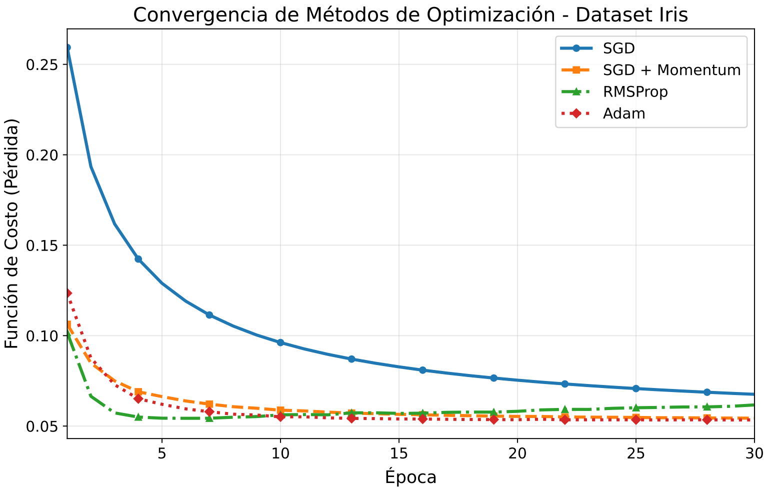
- Adam, RMSProp
- Aproximan información de segundo orden
- Escalables a problemas masivos
- $O(n)$ memoria

Conexión Clave

Adam puede verse como una aproximación diagonal de métodos Quasi-Newton:

$$\text{Adam} \approx \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathrm{diag}(\mathrm{H}^{-1})\nabla J(\boldsymbol{\theta}_t)$$

Curvas de Convergencia



- **Adam**: Convergencia más rápida (5 épocas)
- **RMSProp**: Buena velocidad, algo oscilante
- **Momentum**: Descenso inicial rápido pero inestable

Análisis Detallado de Resultados

Adam - El Ganador

- Pérdida final: $\sim 0{,}12$ en solo 10 épocas
- Curva suave y estable
- Mínima necesidad de ajuste manual

Momentum - Doble Filo

- Descenso inicial más drástico (época 2: pérdida $\sim 0{,}1$)
- Pero oscilaciones significativas después
- Evidencia del problema de "sobrepaso"

RMSProp vs SGD

- RMSProp: Convergencia acelerada ($\sim 0{,}15$ final)
- SGD: Lento pero confiable ($\sim 0{,}30$ a época 30)

Métricas de Rendimiento Final

Algoritmo	Costo Final	Precisión Train	Precisión Test
SGD	0.067	96.25 %	95.00 %
SGD + Momentum	0.054	96.25 %	95.00 %
RMSProp	0.062	96.25 %	90.00 %
Adam	0.053	97.50 %	95.00 %

Observaciones Importantes

- Precisión final similar en todos los métodos
- Diferencias principales en **velocidad de convergencia**
- Adam ligeramente superior en precisión de entrenamiento

Ventajas y Desventajas por Método

SGD

Pros:

- Simple de implementar
- Estable y confiable
- Buena generalización

Cons:

- Convergencia lenta
- Sensible a tasa de aprendizaje

Momentum

Pros:

- Acelera convergencia inicial
- Supera valles estrechos

Cons:

- Puede oscilar mucho

RMSProp

Pros:

- Adaptación automática
- Maneja bien gradientes dispersos

Cons:

- Algo más complejo
- Requiere ajuste de ρ

Adam

Pros:

- Mejor de ambos mundos
- Funciona .ºut-of-the-box"
- Robusto y rápido

Cons:

- Posible overfitting

Perspectiva del Análisis Numérico

Compromiso Fundamental

Velocidad de convergencia vs. Escalabilidad computacional

Método	Convergencia	Costo/Iter	Memoria
Newton-Raphson	Cuadrática	$O(n^3)$	$O(n^2)$
Quasi-Newton	Superlineal	$O(n^2)$	$O(n^2)$
Adam	Lineal	$O(n)$	$O(n)$
SGD	Lineal	$O(n)$	$O(n)$

Insight Clave

Los métodos adaptativos modernos (Adam, RMSProp) aproximan información de segundo orden con costo de primer orden

Conclusiones Principales

1. **Adam es el claro ganador** para convergencia rápida y facilidad de uso
2. **Momentum acelera** pero requiere cuidado en la calibración
3. **RMSProp** ofrece buen compromiso entre velocidad y estabilidad
4. **SGD básico** sigue siendo válido para casos que priorizan generalización

Desde el Análisis Numérico

Los métodos estudiados representan diferentes estrategias para incorporar información de curvatura (segundo orden) manteniendo la escalabilidad computacional

Recomendación Práctica

- **Para empezar**: Adam con parámetros por defecto
- **Para ajuste fino**: Considerar híbrido (Adam inicial + SGD final)
- **Para datos grandes**: RMSProp o Adam
- **Para mejor generalización**: SGD con momentum

El Futuro de la Optimización

Tendencias Actuales

- **Métodos híbridos**: Combinando lo mejor de diferentes enfoques
- **Optimización automática**: Learning rate schedules adaptativos
- **Aproximaciones de segundo orden**: Métodos quasi-Newton escalables
- **Optimización distribuida**: Para modelos masivos como GPT

Aplicaciones Emergentes

- **Federated Learning**: Optimización distribuida sin centralizar datos
- **Neural Architecture Search**: Optimización de arquitecturas
- **Meta-learning**: Aprender a optimizar

Mensaje final: La optimización es el corazón de la IA moderna

Mensajes Clave

No existe un optimizador universal

- La elección depende del problema específico
- Adam es un excelente punto de partida
- Siempre monitorear tanto entrenamiento como validación
- La implementación correcta es tan importante como la elección del algoritmo

El entendimiento teórico guía las decisiones prácticas

¿Preguntas?

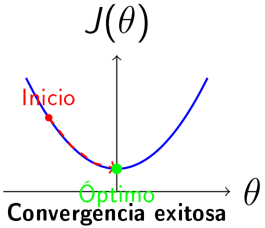
Gracias por su atención

Brandon Trigueros
brandon.trigueros@ucr.ac.cr

Código disponible en:
https://github.com/BrandonTrigueros/gradient-descent-research

¡Gracias por su atención!

¿Preguntas?



"En optimización, como en la vida, el camino más corto no siempre es el más eficiente"

Respaldo: Fórmulas Detalladas de Adam

Algoritmo Completo

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t \tag{8}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)g_t^2 \tag{9}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \text{(corrección de sesgo)} \tag{10}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \quad \text{(corrección de sesgo)} \tag{11}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\varepsilon}\hat{m}_t \tag{12}$$

- m_t: estimación del primer momento (media)
- v_t: estimación del segundo momento (varianza no centrada)
- Las correcciones de sesgo son importantes en las primeras iteraciones

Respaldo: Datos del Experimento

Época	SGD	Momentum	RMSProp	Adam
1	0.259	0.106	0.102	0.124
2	0.193	0.085	0.067	0.088
5	0.129	0.066	0.054	0.062
10	0.099	0.058	0.055	0.054
20	0.078	0.053	0.060	0.053
30	0.068	0.054	0.062	0.053

Cuadro: Evolución del costo de entrenamiento

- Adam converge más rápido en las primeras épocas
- Momentum muestra la mayor reducción inicial pero luego oscila
- SGD mejora de manera más gradual y consistente