MEMAD-T05

ALEJANDRO ZARATE MACIAS

22 de Septiembre 2025

Introducción

El objetivo de esta tarea es implementar métodos estocásticos de optimización y compararlos con métodos deterministas previamente utilizados en tareas anteriores. Se busca evaluar el desempeño de diferentes algoritmos aplicados a problemas de regresión polinomial, implementando:

- SGD
- Minibatch
- ADAM
- Comparación con Gauss-Newton (método determinista de la tarea T04)

A través de estos problemas se realizarán comparativas entre los distintos algoritmos para identificar sus ventajas y desventajas en términos de precisión, eficiencia computacional y robustez ante diferentes condiciones del problema.

1 Problema 1

1.1 Enunciado

Considere la siguiente función

$$f(x) = 2^{cos(x^2)}, \quad x \in (-\pi, \pi).$$

A continuación, haga lo siguiente:

- Calcule 500 pares de datos de muestra $D = x_i, f(x_i)$ donde los x_i están equiespaciados.
- Realice un gráfico de D.
- Considere un modelo polinomial de grado $\mathcal{N}$:

$$h(x_i, \theta) = \hat{y_i} = \sum_{\ell=0}^{\mathcal{N}} \theta_\ell x_i^\ell, \quad \theta = (\theta_0, \theta_1, \theta_2, ..., \theta_{\mathcal{N}}) \in \mathbb{R}^{\mathcal{N}+1}.$$

- Considere una función de pérdida y costo de MSE:

$$Loss(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2,$$
$$Cost(\theta; D) = \frac{1}{m} \sum_{i=1}^{m} Loss(y_i, \hat{y}_i)$$

- Codifique un script en Python para resolver el problema de optimización asociado utilizando un algoritmo de búsqueda lineal determinista. Puede probar $\mathcal{N} = 8$, por ejemplo.

- Haz una gráfica de la solución contra $f(x_i)$ y muestra en otra gráfica cómo el costo disminuye a medida que pasan las iteraciones.

1.2 Metodología

Para resolver este problema, primero se deben implementar las funciones definidas en el enunciado ($f(x)$, h, *loss* y *cost*).

Una vez implementadas estas funciones, procederemos a utilizar un algoritmo de búsqueda lineal determinista. Para este caso, emplearemos el método de Gauss-Newton implementado en tarea 04, pero ya con las correcciones realizadas en el algoritmo, a diferencia de la entrega pasada (en mi caso).

El procedimiento consistirá en:

1. Generar 500 puntos equiespaciados en el intervalo $(-\pi, \pi)$ utilizando numpy y la función "linespace".

2. Calcular los valores de $f(x_i)$ para cada punto.

3. Aplicar el algoritmo de Gauss-Newton para encontrar los parámetros óptimos θ.

4. Finalmente graficar los resultados y el comportamiento de la función de costo.

1.3 Resultados

La Figura 1 muestra la función original $f(x) = 2^{\cos(x^2)}$ evaluada en los 500 puntos de muestra equiespaciados en el intervalo $(-\pi, \pi)$.

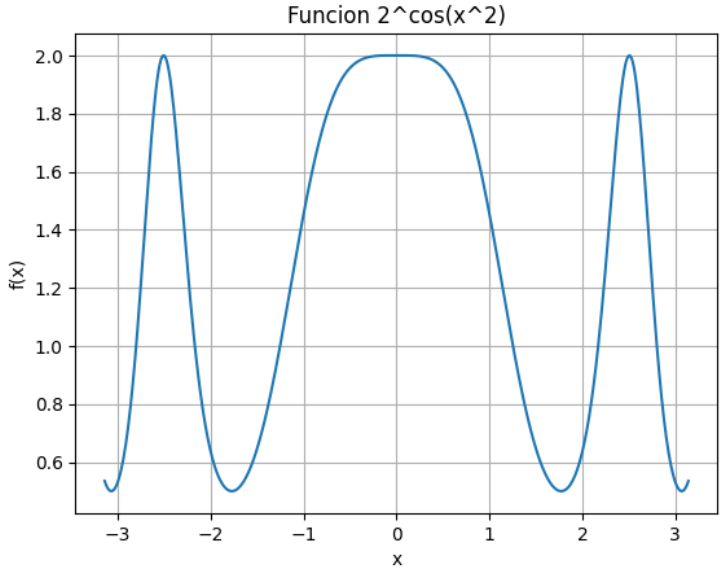

Figure 1: Función original $f(x) = 2^{\cos(x^2)}$ con 500 puntos de muestra

Para los parámetros de optimización: $\mathcal{N} = 8$, tasa de aprendizaje inicial $\alpha = 0.01$, 2000 iteraciones máximas y tolerancia de 1×10^{-8}, obtuvimos que el algoritmo de Gauss-Newton alcanzó su punto óptimo en 170 iteraciones.

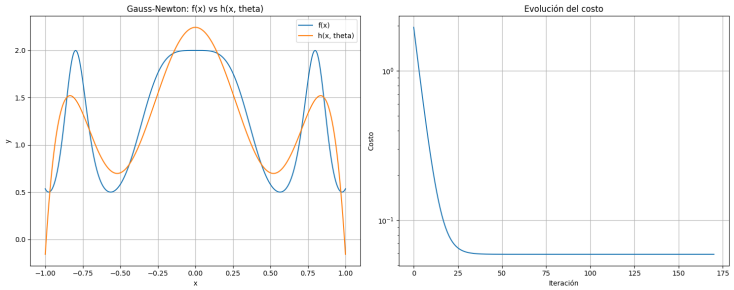

Figure 2: Resultados del ajuste polinomial usando Gauss-Newton con $\mathcal{N} = 8$

1.4 Discusión

Después de haber podido implementar de manera exitosa el método de Gauss-Newton para esta tarea, a diferencia de la implementación pasada, logramos la convergencia de la función de manera satisfactoria.

Para el caso del polinomio de grado $\mathcal{N} = 8$, aunque la función resultante no hace un "fit" exacto a la función original, se asemeja considerablemente a ella.

Esto es esperado dado que estamos aproximando una función con un polinomio de grado relativamente bajo a comparacion de la original ($f(x)$).

La convergencia en 170 iteraciones indica que el algoritmo fue eficiente para este problema particular, y la tolerancia establecida fue adecuada para obtener una solución de buena calidad.

1.5 Conclusión

Logramos resolver exitosamente el problema de optimización utilizando el método de Gauss-Newton. La implementación mejorada del algoritmo nos permitió obtener una convergencia adecuada y un ajuste polinomial que aproxima razonablemente bien la función objetivo $f(x) = 2^{\cos(x^2)}$. Los resultados demuestran que el enfoque determinista de búsqueda lineal es efectivo para este tipo de problemas de regresión polinomial, logrando una solución satisfactoria en un número bajo de iteraciones.

2 Problema 2

2.1 Enunciado

Intenta encontrar un tamaño más adecuado para el modelo paramétrico del problema anterior resolviendo de nuevo varios valores de $\mathcal{N}$. Crea gráficos que muestren tus hallazgos. A partir de aquí, puedes fijar este valor de N para los problemas restantes.

2.2 Metodología

Para encontrar el tamaño más adecuado del modelo paramétrico, tomaremos la implementación previa del algoritmo de Gauss-Newton y procederemos a iterar con distintos valores de $\mathcal{N}$ para determinar cuál puede ser el más óptimo para este problema.

Después de varias pruebas experimentales, se descubrió que los puntos donde la función resultante llega a adquirir diferentes formas es con los valores $\mathcal{N} = [2, 4, 8, 12, 16, 20]$. Estos valores nos permitirán observar cómo evoluciona la aproximación polinomial conforme aumenta el grado.

El procedimiento consistirá en:

1. Aplicar el algoritmo de Gauss-Newton para cada valor de $\mathcal{N}$
2. Agrupar y mostrar las gráficas del resultado de la optimización de los valores θ
3. Comparar el avance del costo para cada grado de polinomio

2.3 Resultados

Para los valores de $\mathcal{N} = [2, 4, 8, 12, 16, 20]$ obtuvimos que casi todas las optimizaciones convergen en un número similar de iteraciones, pero los resultados son considerablemente distintos entre sí.

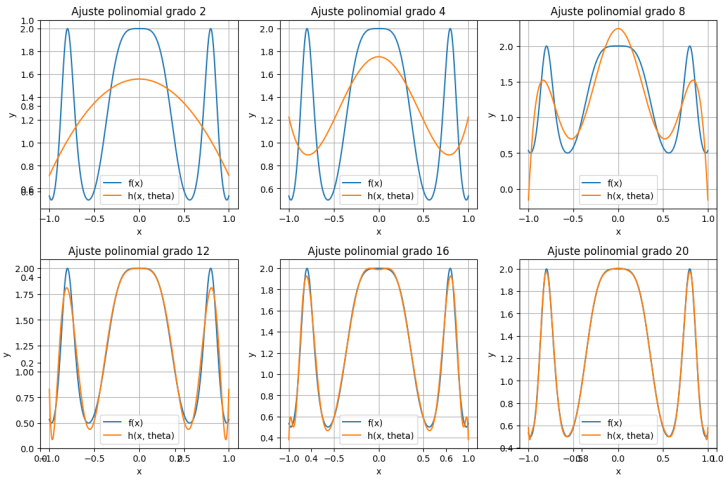

Figure 3: Comparación de ajustes polinomiales para diferentes grados $\mathcal{N}$

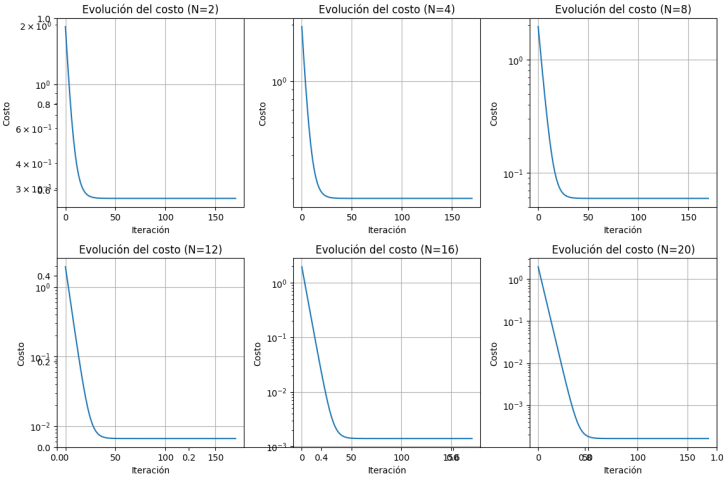

Figure 4: Evolución del costo para diferentes grados de polinomio

Los resultados muestran que conforme aumenta el grado del polinomio, la aproximación a la función original mejora significativamente, especialmente para $\mathcal{N} \geq 12$.

2.4 Discusión

Pese a que $\mathcal{N} = 20$ resultó ser el valor más apegado a la función original, es considerablemente más costoso computacionalmente calcularlo. Por el contrario, $\mathcal{N} = 12$ representa un polinomio no tan grande pero con una aproximación bastante decente.

Los polinomios de grado bajo ($\mathcal{N} = 2, 4$) muestran aproximaciones muy limitadas que no capturan la complejidad de la función original. Los grados intermedios ($\mathcal{N} = 8, 12$) ofrecen un buen balance entre precisión y eficiencia computacional, mientras que los grados altos ($\mathcal{N} = 16, 20$) proporcionan la mejor aproximación a costa de mayor complejidad.

2.5 Conclusión

Logramos identificar que $\mathcal{N} = 12$ representa el valor óptimo para el modelo paramétrico, ofreciendo un equilibrio adecuado entre precisión de aproximación y eficiencia computacional. Pese a lo anteriormente mencionado, para los siguientes problemas se estará utilizando $\mathcal{N} = 20$ simplemente para comparar los siguientes algoritmos en un terreno más demandante e intentar exprimir sus ventajas lo más que se pueda.

3 Problema 3

3.1 Enunciado

Cree un script para resolver el Problema 1 utilizando un gradiente descendente de Minibach con una tasa de aprendizaje constante α. Genere una gráfica de la solución en función de $f(x_i)$ y muestre en otra gráfica cómo disminuye el coste a medida que transcurren las iteraciones.

3.2 Metodología

Para resolver este problema utilizaremos el algoritmo de descenso de gradiente estocástico con enfoque en Minibatch como alternativa al método determinista de Gauss-Newton empleado anteriormente para reducir un poco la carga computacional sobre las iteraciones.

Implementaremos las mismas funciones base ($f(x)$, h, *loss* y *cost*) junto con una función de gradiente específica para SGD, ademas de una función enfocada a dividir nuestro dataset en "batches" para poder iterar sobre ellos.

El procedimiento consistirá en:

1. Utilizar los mismos 500 puntos equiespaciados del Problema 1
2. Implementar el algoritmo SGD Minibatch con tamaño de lote de 50 muestras
3. Aplicar una tasa de aprendizaje constante $\alpha = 0.1$

4. Usar $\mathcal{N} = 20$ (basado en los resultados del Problema 2)
5. Ejecutar el algoritmo por 2000 épocas máximo
6. Graficar los resultados y la evolución del costo

3.3 Resultados

Para los parámetros establecidos: $\mathcal{N} = 20$, $\alpha = 0.1$, tamaño de lote = 50, y 2000 épocas máximas, obtuvimos una convergencia satisfactoria del algoritmo SGD Minibatch.

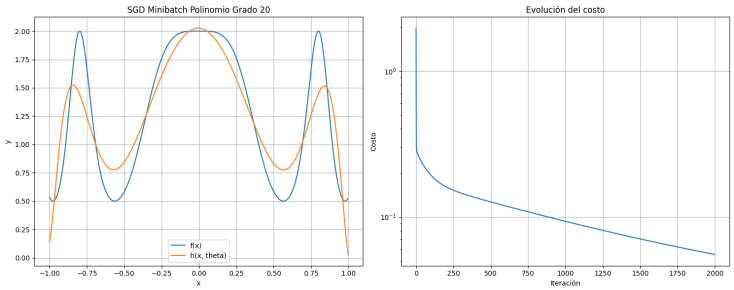

Figure 5: Resultados del ajuste polinomial usando SGD Minibatch con $\mathcal{N} = 20$

3.4 Discusión

El método SGD Minibatch demostró ser efectivo para este problema de regresión polinomial. A diferencia del método determinista de Gauss-Newton, SGD no aproxima tan bien la función como en el caso de Gauss Newton, e incluso hizo uso de todas las iteraciones indicadas sin alcanzar la tolerancia indicada. Pese a esto, los resultados son bastante buenos y en velocidad sí se percibe que, pese al hacer más iteraciones, SGD es bastante más rápido que Gauss-Newton.

La tasa de aprendizaje de 0.1 resultó adecuada para lograr convergencia sin causar inestabilidad en el entrenamiento, y el tamaño de lote de 50 proporcionó un buen balance entre eficiencia computacional y estabilidad.

3.5 Conclusión

Logramos implementar exitosamente el algoritmo SGD Minibatch para resolver el mismo problema de aproximación polinomial. El método estocástico proporcionó resultados comparables al método determinista, demostrando la viabilidad de enfoques estocásticos para problemas de regresión.

4 Problema 4

4.1 Enunciado

Repita el Problema 3 usando Adam.

4.2 Metodología

Para este problema implementaremos el algoritmo ADAM, el perfecto balance entre el manejo de datos de minibatch, momento y RMSprop para el ajuste de theta. Los pasos serán los siguientes:

1. Utilizar los mismos 500 puntos equiespaciados de problemas anteriores
2. Implementar ADAM con parámetros estándar: $\beta_1 = 0.9$, $\beta_2 = 0.999$
3. Aplicar $\alpha = 0.1$, $\mathcal{N} = 20$, tamaño de lote $= 50$, 2000 épocas máximo
4. Comparar resultados con SGD Minibatch

4.3 Resultados

Para los parámetros establecidos obtuvimos una convergencia notablemente superior con ADAM comparado con SGD Minibatch.

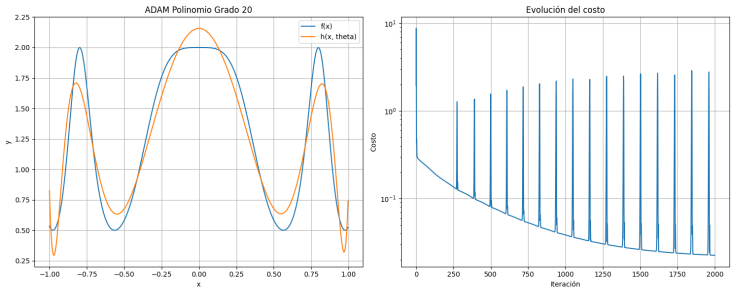

Figure 6: Resultados del ajuste polinomial usando ADAM con $\mathcal{N} = 20$

4.4 Discusión

ADAM demostró ventajas claras sobre SGD Minibatch en términos de calidad del ajuste final y mejor aproximación a la función original. Sin embargo, mostró un comportamiento interesante donde la función de costo presenta picos iniciales que luego se corrigen, contrastando con SGD que mantuvo una evolución más constante sin picos pronunciados.

4.5 Conclusión

Logramos implementar exitosamente ADAM obteniendo resultados superiores a SGD Minibatch en estabilidad y calidad de convergencia. Los resultados confirman las ventajas de los métodos adaptativos sobre gradiente tradicional, estableciendo una base sólida para las comparaciones del siguiente problema.

5 Problema 5

5.1 Enunciado

Compara las soluciones y el rendimiento de los tres algoritmos que utilizaste para resolver el problema 1. Escribe tus conclusiones y haz algunos gráficos.

5.2 Metodología

Para este problema reutilizaremos todas las clases ya implementadas (Gauss-Newton, SGD y Adam) para comparar los resultados de los tres algoritmos en una misma gráfica y evaluar sus diferencias en términos de calidad de ajuste y comportamiento de convergencia.

5.3 Resultados

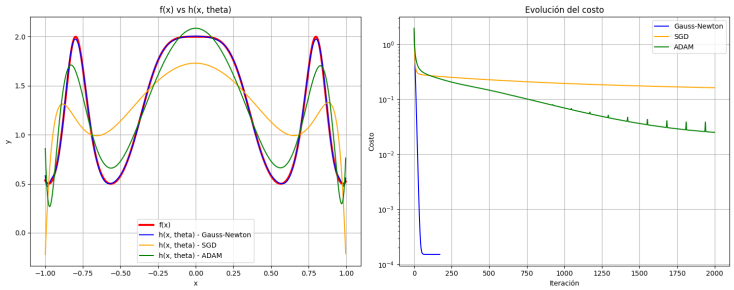

Figure 7: Comparación de los tres algoritmos: Gauss-Newton, SGD Minibatch y ADAM

5.4 Discusión

Como hemos observado desde el inicio, Gauss-Newton representa el mejor ajuste a la función original a costa de un mayor costo computacional. Por el contrario, SGD y ADAM son considerablemente más rápidos, aunque generalizan más el resultado sin parecerse completamente a la función original.

Entre los métodos, ADAM se destaca al poder disminuir más la función de costo que SGD y lograr un mejor parecido a la función original. Esto confirma las ventajas de los métodos adaptativos en términos de calidad de convergencia, manteniendo la eficiencia computacional.

5.5 Conclusión

La comparación confirma que existe un balance entre precisión y eficiencia computacional. Gauss-Newton ofrece la mayor precisión pero con mayor costo, mientras que los métodos estocásticos proporcionan soluciones más rápidas con calidad aceptable. ADAM destaca como el mejor entre ambos enfoques, combinando eficiencia computacional con mejor calidad de ajuste que SGD tradicional.

6 Problema 6

6.1 Enunciado

Hasta ahora has resuelto un problema sin ruido, es decir,

$$f(x) = 2^{cos(x^2)} + \mathcal{X}, \quad x \in (-\pi, \pi), \mathcal{X} \sim \mathcal{N}(0, \sigma^2),$$

Donde solo se ha considerado el caso con $\sigma^2 = 0$. Repita los ejercicios 1 a 5 para los niveles de ruido $\sigma^2 = 0.05, 0.1, 0.5$. No olvide anotar todas sus conclusiones y reflexiones.

6.2 Metodología

Para evaluar el comportamiento de los algoritmos bajo condiciones de ruido, implementamos una nueva función "f_noise(x)" que altera los valores de y en función de x y el valor de σ indicado.

Aplicamos los tres algoritmos (Gauss-Newton, SGD Minibatch y ADAM) para cada nivel de ruido propuesto: $\sigma^2 = 0.05, 0.1, 0.5$, mediante un loop que compare todos los métodos según σ^2, pero en este caso se optará por utilizar $\mathcal{N} = 12$ para disminuir un poco la carga computacional de estos ciclos iterativos donde debemos evalur muchas veces los algoritmos.

6.3 Resultados

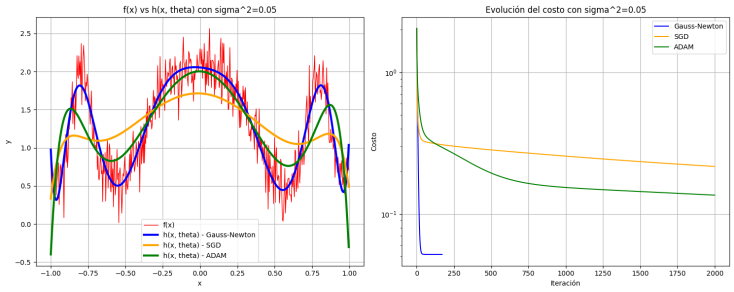

Figure 8: Comparación de algoritmos con ruido $\sigma^2 = 0.05$

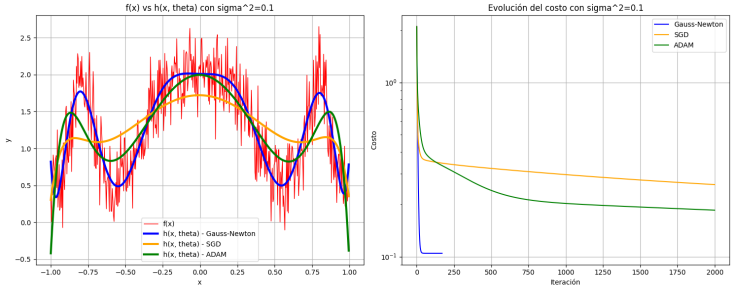

Figure 9: Comparación de algoritmos con ruido $\sigma^2 = 0.1$

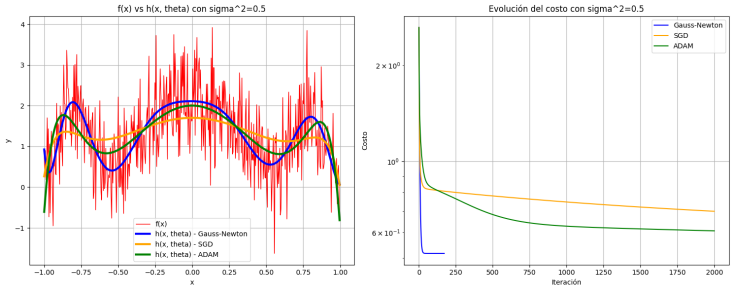

Figure 10: Comparación de algoritmos con ruido $\sigma^2 = 0.5$

6.4 Discusión

Los resultados muestran que conforme aumenta el nivel de ruido, todos los algoritmos enfrentan mayor dificultad para aproximar la función original. Sin embargo, mantienen patrones de comportamiento consistentes con las observaciones previas.

Gauss-Newton continúa proporcionando el mejor ajuste incluso en presencia de ruido, aunque su ventaja se reduce conforme aumenta σ^2. Los métodos estocásticos (SGD y ADAM) muestran mayor robustez al ruido, manteniendo aproximaciones razonables incluso con $\sigma^2 = 0.5$, siendo ADAM nuevamente quien más destaca.

6.5 Conclusión

La evaluación con ruido confirma que los algoritmos mantienen sus características distintivas bajo condiciones mas complejas. Gauss-Newton conserva su precisión superior, mientras que los métodos estocásticos ofrecen mayor robustez. El ruido afecta la calidad del ajuste en todos los casos, pero ADAM vuelve a mostrarse como el método más equilibrado para escenarios con datos ruidosos.

7 Problema 7

7.1 Enunciado

Explique con sus propias palabras lo siguiente:

(a) Las diferencias y similitudes entre el descenso de gradiente estocástico, el descenso más pronunciado y el descenso de gradiente en minilotes.

(b) Algunos ejemplos del uso de los tres últimos algoritmos citados (es decir, explique cómo identificar cuándo se debe utilizar un algoritmo entre otros).

(c) Algunos ejemplos del uso de un enfoque determinista (Newton, descenso más pronunciado, BFGS, Gauss-Newton, etc.) y un enfoque estocástico (descenso de gradiente estocástico, descenso de gradiente en minilotes, Adam, etc.), es decir, explique cómo identificar cuándo se debe utilizar un enfoque particular sobre otro.

7.2 Análisis Personal

(a) Diferencias y similitudes entre SGD, descenso más pronunciado y minibatch:

Desde mi experiencia a lo largo de este primer modulo, estos tres métodos han sido los más simples de implementar, principalmenta para el caso de steepest descent cuando nuestro α es de valor fijo. Dan buenos resultados, y en ambos casos los gradientes podemos depender de fórmulas más generales de implementar sin importar el problema.

Para steepest descent tenemos la posibilidad de usar diferencias centrales, y en el caso de los estocásticos nos enfocamos en optimizar la función de costo, la cual ya tiene una forma dada para cada problema. Como comparativa entre estocásticos, SGD y minibatch son lo mismo en implementación para los ajustes de θ, difiriendo solamente en la cantidad de datos que se procesan, ya que minibatch nos permite dar ajustes con valores más agrupados según la necesidad y capacidad computacional, por lo que es mejor priorizar su implementación sobre SGD (aunque en esencia son lo mismo).

La mayor desventaja es que steepest computacionalmente es más costoso que los estocásticos, como lo hemos venido recalcando en esta tarea, a menos que usemos gradientes analíticos. Pero para ciertos casos puede ser igual de simple de implementar y más eficiente. Sin embargo, para la mayoría de los casos, sobre todo en deep learning, los estocásticos son siempre la mejor opción.

(b) Ejemplos de uso y cuándo elegir cada algoritmo:

Reiterando lo anteriormente mencionado en el inciso (a), si el problema a optimizar no es tan complejo y podemos calcular el gradiente de manera analítica, es mejor optar por los métodos deterministas como steepest descent al dar resultados más precisos y en un número razonable de iteraciones. Como lo veíamos en el caso de funciones como esfera trasladada vista en tareas pasadas, que al ser de orden cuadrático, los resultados de la optimización eran casi inmediatos.

Desgraciadamente no podemos vivir siempre en el mundo ideal, y tenemos el ejemplo de funciones vistas como Rosenbrook y Perm, por lo que cuando las funciones se vuelven cada vez más complejas, es mucho mejor optar por la facilidad de los estocásticos y principalmente por minibatch si es que nuestro volumen de datos es muy grande. En estos casos, la capacidad de procesar datos en lotes (ya sea pequeños o grandes, depende del poder computacional que tengamos) nos permite manejar datasets masivos sin saturar la memoria del sistema, algo que sería imposible con métodos deterministas que requieren procesar todo el conjunto de datos en cada iteración.

(c) Enfoques deterministas vs estocásticos - cuándo usar cada uno:

Métodos como Newton, Quasi-Newton (BFGS) o Gauss-Newton son mucho más complejos de implementar a nivel de código, pero los resultados valen totalmente la pena al ser mucho mejores que steepest descent en el caso de modelos deterministas. Mientras que la complejidad de los estocásticos no aumenta tanto en comparación al aumento de mejora en resultados que nos dan en comparación de SGD a ADAM, donde por simplemente un par de lineas de codigo extras, los resultados mejoran considerablemente.

Si queremos resultados precisos sobre cuándo las funciones alcanzan sus puntos críticos y dónde es que estas son óptimas sin importar la complejidad computacional y sacrificando tiempos de ejecución (y si es que nos lo podemos permitir en temas de recursos de cpu/ram/gpu), lo ideal es usar deterministas por la eficacia de sus resultados.

Ahora bien, métodos como ADAM resultaron ser muy óptimos para resolver estos problemas como lo veíamos en el caso del polinomio $N = 20$, que pese a no ser un valor exacto, nos da una buena generalización, que es lo que esencialmente buscamos en problemas como regresión. Queremos que nuestro modelo tome una forma bastante similar a como están distribuidos los datos sin llegar a hacer overfitting, lo cual nos ayuda para predecir otros valores. Es por esto que estos predominan completamente el área del deep learning.

7.3 Conclusión

En resumen, ambos tipos de algoritmos tienen sus enfoques particulares y podemos disponer de ambos según el caso específico que enfrentemos. Los métodos deterministas ofrecen precisión superior cuando necesitamos resultados exactos y tenemos los recursos computacionales disponibles, mientras que los métodos estocásticos proporcionan eficiencia y escalabilidad para problemas más complejos y datasets grandes.

Sin embargo, en mi experiencia personal, prefiero más los estocásticos por su simplicidad de implementación y buenos resultados. La facilidad para programarlos, combinada con su capacidad de manejar datos masivos y su robustez en diferentes tipos de problemas, los convierte en una herramienta más práctica y versátil para la mayoría de situaciones que he enfrentado en este curso.