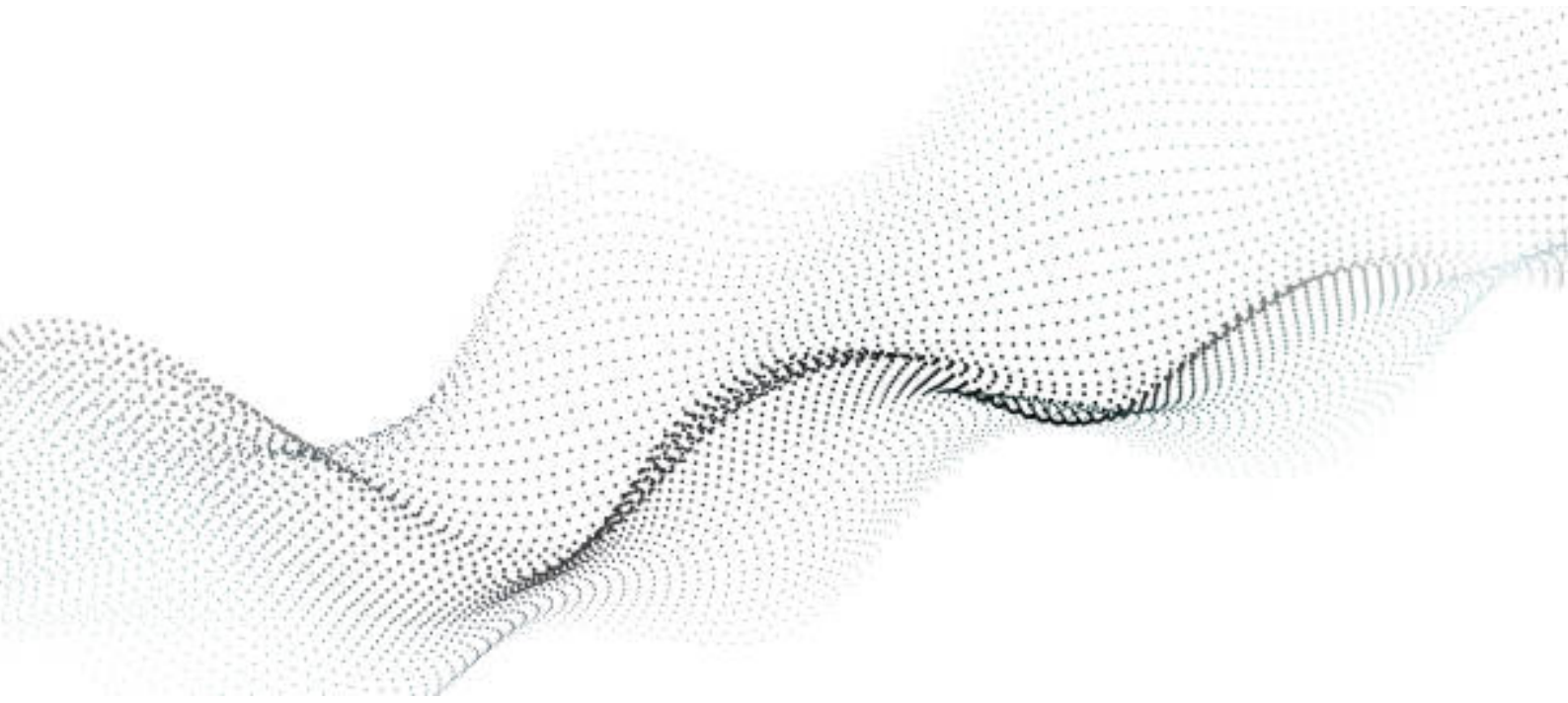

Instituto Tecnológico Autónomo de México

Diplomado: Ciencia de Datos y Machine Learning Aplicado a Finanzas

Módulo 3: Ciencia de Datos

Tema

Descenso del Gradiente

Nombre: Denzel Arik Martinez Maldonado

Al entrenar un modelo puede parecer tan sencillo como "meter datos" y esperar que el algoritmo aprenda, pero en realidad el entrenamiento es un proceso delicado. Esto se debe a que entrenar un modelo implica ajustar parámetros taambien conocidos como pesos para que el error del modelo sea lo más pequeño posible y ese ajuste no se hace de golpe sino por pasos. A este proceso de ir ajustando paso a paso se le conoce como **descenso del gradiente**.

Imaginemos que el error del modelo es como una montaña y el objetivo es llegar al punto más bajo. El gradiente es como una brújula que nos dice hacia dónde está subiendo más la montaña, por lo que avanzar en sentido contrario nos hace descender. Sin embargo, el verdadero reto está en cuánto avanzamos en cada paso. El "tamaño del paso" se controla con un parámetro llamado **learning rate** que es la tasa de aprendizaje. Si el learning rate es muy grande, el modelo puede dar pasos tan grandes que se pase del mínimo y empiece a rebotar o incluso empeorar. Si el learning rate es muy pequeño, el modelo puede tardar demasiado en aprender y el entrenamiento se vuelve lento.

Además, en la práctica no siempre se puede calcular la dirección óptima usando todos los datos al mismo tiempo, ya sea por tamaño del dataset o por costo computacional. Por esta razón existen distintas formas de aplicar el descenso por gradiente dependiendo de cuánta información se use en cada actualización:

- **Batch Gradient Descent**
  En este enfoque se usan todos los datos para calcular cada actualización. Esto hace que el paso sea más estable, pero también lo vuelve muy lento cuando el dataset es grande. En términos prácticos, sería como querer tomar una decisión solo después de revisar absolutamente toda la información disponible cada vez.

- **Stochastic Gradient Descent (SGD)**
  Aquí se actualiza el modelo usando un solo dato a la vez. La ventaja es que el entrenamiento se vuelve más rápido y permite aprender en escenarios más grandes, pero la desventaja es que el camino hacia el mínimo se vuelve ruidoso, como cada dato puede ser diferente el modelo puede avanzar de manera inestable con fluctuaciones constantes.

- **Mini-batch Gradient Descent**
  Este enfoque es una combinación de los dos anteriores. En lugar de usar todos los datos o solo uno, se usan pequeños grupos de datos, mini-batches. En la práctica es el enfoque más común porque suele ser un buen balance entre velocidad y estabilidad.

Aunque el mini-batch es el estándar, el entrenamiento todavía enfrenta problemas importantes. Uno de los más comunes es que el descenso puede volverse un movimiento de **zigzag**, donde el modelo avanza en una dirección pero rebota demasiado en otra. Esto sucede mucho cuando el error tiene forma de valle largo y estrecho, donde en un eje la pendiente es muy fuerte y en otro es muy suave. Otro problema es que usar una sola tasa de aprendizaje para todos los parámetros puede ser ineficiente, ya que no todos los parámetros se comportan igual ni necesitan ajustes del mismo tamaño.

Para atender estos problemas surgen algoritmos de optimización que en esencia son mejoras al descenso por gradiente tradicional diseñadas para que el entrenamiento sea más estable, más rápido y menos sensible a elegir manualmente el learning rate.

- **Momentum**
  Momentum introduce la idea de "inercia" en el aprendizaje. En lugar de depender únicamente del paso actual guarda parte de la dirección anterior y la combina con la actual. Esto hace que, si el modelo ha estado avanzando consistentemente hacia una dirección, siga avanzando con más fuerza, y si estaba rebotando, el movimiento

se suavice. En pocas palabras, el momentum ayuda a reducir el zigzag y a acelerar el avance cuando la dirección es consistente.

- **RMSprop**
  RMSprop busca resolver un problema distinto: que una sola tasa de aprendizaje no se adapta bien a todos los parámetros. RMSprop ajusta automáticamente el tamaño del paso dependiendo del comportamiento reciente del gradiente. Así cuando el gradiente es grande e inestable, reduce el paso; y cuando el gradiente es pequeño, permite avanzar sin perder eficiencia. Esto lo vuelve una opción muy útil cuando el entrenamiento es inestable.

- **Adam**
  Adam puede verse como una combinación práctica entre momentum y RMSprop. Por un lado mantiene memoria de la dirección como lo hace el momentum y por otro ajusta automáticamente el tamaño del paso por parámetro como lo hace RMSprop. Por eso es uno de los optimizadores más usados en la práctica ya que suele funcionar bien sin necesidad de demasiada calibración manual.

Es importante entender que no existe un optimizador perfecto para todos los casos. En la práctica, Adam suele ser una primera opción por su estabilidad y rapidez, mientras que momentum puede ser muy útil cuando el entrenamiento se mueve en zigzag. RMSprop y otros métodos adaptativos suelen ser adecuados cuando se busca que la tasa de aprendizaje se ajuste automáticamente. Mas allá de memorizar fórmulas el punto central es comprender que entrenar modelos no es solo aplicar un algoritmo, sino controlar cómo aprende: qué tan rápido, qué tan estable y con qué tanto ruido avanza hacia una solución.

Otro punto importante en la práctica es que cuando los datos o el modelo son muy grandes el problema ya no es solo elegir un buen optimizador, sino hacer que el entrenamiento sea viable en tiempo. Para esto se recurre a la **paralelización**, que consiste en entrenar usando varios procesadores o varias máquinas al mismo tiempo. La idea general es dividir el trabajo para calcular gradientes más rápido, pero esto también introduce nuevos retos.
Si varias partes del sistema calculan actualizaciones al mismo tiempo, pueden ocurrir **desfases**: algunos procesos están actualizando parámetros con información más "vieja" que otros. Esto puede hacer que el entrenamiento sea más rápido, pero no necesariamente más estable. En otras palabras, paralelizar suele ser un intercambio entre velocidad y calidad de convergencia.

Existen estrategias donde todos los procesos se sincronizan en cada actualización proprocioandno más estabilidad, pero más lento por la espera y otras donde se permite un entrenamiento asíncrono más rápido, pero con riesgo de que las actualizaciones se estorben o se contradigan. Bajo ciertas condiciones, sobre todo cuando los gradientes son esparsos o los parámetros que se actualizan no se traslapan tanto, los métodos asíncronos pueden funcionar muy bien y acelerar el entrenamiento sin afectar demasiado el resultado. En general la paralelización es una herramienta clave cuando el volumen de datos o el tamaño del modelo crece, pero requiere cuidar la coordinación de las actualizaciones para que el modelo no aprenda de forma errática.