# Tarea corta **01**

Jason Carmona (*200312094*)
Edisson López (*2013103311*)

Marzo 19, 2021

**Descripción de la tarea**: Tomar el código de Linear Regression visto en clase, y convertirlo en dos versiones: (1) usando solo listas (Python a pie), y (2) usando Pytorch. Debe obtener los mismos resultados con los set de datos datos en clase. Con base a eso se crea éste documento con las **diferencias entre uso de Listas, Numpy (original) y Pytorch**.

## Diferencias

### Numpy vs Listas (Python a pie)

A continuación, se listan las principales diferencias encontradas, enfocadas en el proceso de programación, obtención de resultados, optimización del código una vez llegado a una solución aceptable, entre otros:

- Rendimiento: La diferencias más marcada se observo en el rendimiento de la aplicación a la hora de hacer uso de rutinas propias de librerias que han sido sumamente refinadas y optimizadas, en comparación a la creación de funciones para cubrir la misma funcionalidad. El uso de recursividad para ajustar el funcionamiento de las operaciones de matrices, a múltiples formas de matrices que se pudieran recibir, cobró un peaje a la hora de ejecutar la aplicación, haciendo qeu cada epoch tomara varios segundos en ejecutarse.

- Facilidad de uso: No es lo mismo definir un vector con la ayuda de una libreria y limitarse a hacer operaciones, dejando que de forma interna se defina la sobrecarga de operadores o se aplique la operación adecuada, sea un valor escalar o vector, que tener que velar por que se estén llamando los métodos que corresponden.

- Variabilidad de casos: A la hora de hacer operaciones con matrices, el uso de librerías externas cubren muchos tipos de estructuras, para ejemplo la multiplicación de una matriz por un vector, teniendo el vector sus valores no en el mismo nivel, sino dentro de matrices anidadas de una sola celda. A pesar de esto, que define de la definición matemática de la operación, la librería es capaz de ejecutar las operaciones. A la hora de tratar de cubrir estos casos en las funciones implementadas, no solamente aumenta la complegidad, sino que también en rendimiento decae, evidenciando aún más lo preferible de utilizar librerías optimizadas para la realización de este tipo de operaciones.

- Claridad y legibilidad: El código cuando hace uso de los vectores facilitados por la librería externa, tiene un enfoque más directo, centrado en la operación necesaria, sin necesidad de especificar llamadas adicionales solo por el tipo de dato con el que se esta haciendo la operación. La definición de las derivadas, pasó de una pocas líneas, a múltiples llamadas a metodos encadenados, lo que resta legibilidad.

- Tendencia a error: Dado el aumento en la cantidad de codigo, la disminución en la legibilidad y el tener que velar por el tipo exacto de dato con el que las operaciones se tienen que realizar, aumenta la tendencia a error. Durante el proceso de desarrollo y posterior a el etablecimiento de equivalencias de funciones programadas según la contraparte de la librería, setuvo que depurar básicamente paso a paso para garantizar que el formato retornado por las funciones creadas, seguían las proporciones esperadas.

- Proceso de optimización: En el caso de las libreridas externas, el proceso de optimización de las operaciones no es un tema que concierne al investigador, por el contrario le permite enfocarse en la mejora de los procesos que se estan ejecutando, el tratamiento de los datos y la eficacia de los mismos. En el caso de trabajo con funciones propias, mucho del proceso de refinamiento se centró en la mejora de las operaciones con matrices, lo que llevó a hacer cambios en las funciones para ajustarse al formato de los datos que se estaba trabajando, removiendo la recursividad y favoreciendo un enfoque más directo, pero perdienddo abstracción.

### Numpy vs Pytorch

A continuación, se listan las principales diferencias encontradas, con base en el estudio del código original de numpy, investigación de primeros pasos con pytorch, ejemplos de numpy comparados con pytorch, documentación de cada función de numpy usada, entre otros:

- Uso de devices: La más notable fue encontrar que se puede definir un device donde pueda correr las instrucciones de pytorch a través de ***torch.device("cpu")*** y ***torch.device("cuda:0")***. Entre los que se encontraron se puede ejecutar en el CPU o sobre CUDA para usar el GPU del computador, con éste segundo se espera un velocidad de hasta 30x más rápido que el CPU. Lamentablemente no pudimos probarlo en nuestras máquinas por falta del hardware necesario.

- Sintaxis: Las instrucciones de numpy y pytorch practicamente la mayoría se le puede hacer un tipo de traducción, es decir cada función de numpy tiene su equivalente para pytorch y en la mayoría de casos es una instrucción por una instrucción, no es necesario pasos intermedios para hacer el cambio de una librería a otra. Se encontró esta referencia en la cual nos apoyamos mucho: PyTorch for Numpy users.

- Optimizaciones: Con Pytorch se puede optimizar el código mucho, mucho más que en Numpy (o al menos en el ejemplo que teníamos), se llegó a ver que en vez de hacer una traducción de pytorch -> numpy tan literal como lo resolvimos, se pueden llegar a implementar (1) tensores con autograd y (2) un módulo nn. Obviamente es una complejidad mayor, pero así el código obtenido es realmente superior.

- Operaciones: Calculando arrays en numpy y tensores en pytorch, se notó un leve mayor rapidez a la hora de generar las operaciones sobre éstos. Por ejemplo con el uso de multiplicaciones de matrices (matmul y multiplicación element-wise (multiply) con mucho más datos e imprimiendo tiempos se logró a notar.