Métodos Computacionales 2023

Trabajo Práctico 2:

Gradiente Descendente

1. Introducción

En términos simples, podemos pensar a las red neuronales como una función. Esta función toma una entrada, la manipula de diversas formas y luego produce una salida. Las manipulaciones específicas que se realizan están predefinidas por la arquitectura de la red, pero los valores (o parámetros) que intervienen en estas manipulaciones se aprenden a partir de los datos. En otras palabras, la red neuronal aprende cómo modificar mejor su entrada para predecir una salida basándose en los datos con los que fue condicionada.

En particular, una red neuronal totalmente conectada (también conocida como "densa") esta compuesta por *capas* en serie que manipulan la entrada. Cada capa de la red toma una entrada $\mathbf{z_i} \in \mathbb{R}^n$, y genera una salida $\mathbf{z_{i+1}} \in \mathbb{R}^m$, donde cada elemento de este último es una combinación lineal de los elementos de $\mathbf{z_i}$ y un término independiente. Esto lo logramos realizando el calculo:

$$\mathbf{z_{i+1}} := W_i \mathbf{z_i} + \mathbf{b_i}, \tag{1}$$

donde $W_i \in \mathbb{R}^{m \times n}$ y $\mathbf{b_i} \in \mathbb{R}^m$. Y es este resultado que se alimenta a la siguiente capa para que pueda realizar su procesamiento y seguir avanzando los resultados, hasta llegar a una capa final la cual, con suerte, genera la información que buscamos.

Es importante notar que, siguiendo este modelo, no podemos expresar funciones no lineales ya que cada capa solo combina linealmente sus entradas. Para que la red sea capaz modelar funciones realmente complejas, además de las operaciones lineales, cada capa aplica también una función no lineal (también llamadas *funciones de activación*) a su resultado antes de propagar a la siguiente capa:

$$\mathbf{z_{i+1}} := \sigma(W_i \mathbf{z_i} + \mathbf{b_i}), \tag{2}$$

donde σ es alguna función no lineal. Típicamente se utilizan la funciones sigmoide[1] o ReLu[2], entre muchas otras. Para operar sobre vectores con múltiples elementos, se aplica la función individualmente a cada elemento.

[1] $\sigma(x) = \dfrac{1}{1 + e^{-x}}$

[2] $\text{ReLu}(x) = \max(0, x)$

2. Definición del problema

En este trabajo modelaremos una red neuronal de dos capas con la siguiente estructura:

$$f_\theta(\mathbf{x}) = W_2\ \sigma(W_1\mathbf{x} + \mathbf{b_1}) + \mathbf{b_2}, \tag{3}$$

es decir, solo tendremos función de activación σ luego de la primer capa. En la última expresión tendremos $W_1 \in \mathbb{R}^{m\times n}$, $b_1 \in \mathbb{R}^{m\times 1}$, $W_2 \in \mathbb{R}^{1\times m}$, $b_2 \in \mathbb{R}$. Además, usaremos la letra θ para nombrar al conjunto de parámetros de nuestra red, es decir: $\theta = (W_1, b_1, W_2, b_2)$.

En un problema de regresión, el objetivo es modelar la relación entre un conjunto de variables independientes o explicativas (también conocidas como características o *features*) y una variable dependiente o de respuesta. Una vez que un modelo se ha ajustado a los datos, se puede utilizar para predecir o estimar la variable dependiente en base a nuevas observaciones de las variables independientes. Formalmente, si tenemos un conjunto de datos *de entrenamiento* $(\mathbf{x_i}, y_i)_{i=1}^N$ donde cada $\mathbf{x_i}$ es un vector de entrada y cada y_i es la salida real correspondiente, queremos encontrar valores para los parámetros (W_1, $\mathbf{b_1}$, W_2, $\mathbf{b_2}$), que minimizan cierta función objetivo. Una función objetivo comúnmente utilizada es la de error cuadrático[3], que mide la diferencia cuadrada entre la salida inferida por la función y la salida esperada. Formalmente:

$$L = \frac{1}{2}\left(f_{\theta_t}(\mathbf{x_i}) - y_i\right)^2. \tag{4}$$

Para lograr minimizar la L vamos a utilizar el método del *gradiente descendente*. A partir de los datos de entrenamiento, evaluaremos el gradiente de la función de pérdida en ese punto con respecto a los parámetros, y luego actualizamos los parámetros en la dirección opuesta al gradiente en ese punto. Este proceso se repite hasta alcanzar convergencia.

Para calcular el gradiente de la función objetivo con respecto a los parámetros de la red, se suele utilizar un algoritmo propuesto por Rumelhart en 1986 llamado *backpropagation*, que consiste en ir calculando el gradiente en pasos sucesivos desde el final hasta el comienzo de la red de forma analítica. Sin embargo, requiere de conocimientos del análisis matricial y tensorial que exceden a la materia. Por lo tanto, nos conformaremos con realizar el computo del *gradiente de forma numérica*, lo cual es más lento y podría llegar a tener mucho más error que el anterior, pero es suficiente en este caso.

Tenemos que calcular el gradiente de la función objetivo en un punto, que equivale a computar las siguientes derivadas parciales respecto a cada elemento de las matrices W y los vectores $\mathbf{b}$:

$$\frac{\partial L}{\partial W_1}\,,\ \frac{\partial L}{\partial \mathbf{b_1}}\,,\ \frac{\partial L}{\partial W_2}\ \text{y}\ \frac{\partial L}{\partial \mathbf{b_2}}. \tag{5}$$

Una vez que calculamos todos estos gradientes, podemos usarlos para actualizar la red. Hacemos esto restando una pequeña cantidad de cada derivada parcial al parámetro correspondiente. Esta cantidad es controlada por un parámetro llamado *learning rate* o tasa de aprendizaje. Es decir, para nuestro espacio de parámetros θ, calculamos:

$$\theta_{t+1} := \theta_t - \alpha\frac{\partial L}{\partial \theta}(\theta_t) \tag{6}$$

Tener en cuenta que, si la tasa de aprendizaje α es demasiado alta, la red puede no converger; si es demasiado baja, la red puede tardar mucho en aprender.

[3]Se le suele agregar un escalado por $\frac{1}{2}$ para que la derivada tenga una expresión más sencilla

2.1. Gradiente numérico

Una estrategia para calcular estas derivadas parciales, consiste en calcular el promedio de los cocientes incrementales a derecha e izquierda. Para obtener la siguiente aproximación, para cada parámetro de la red calculamos:

$$\frac{\partial L}{\partial p} \sim \frac{L(\theta_t, p+\epsilon) - L(\theta_t, p-\epsilon)}{2\epsilon} \tag{7}$$

donde usamos p de forma genérica para referirnos a cada elemento $w^1_{i,j}$, b^1_j, $w^2_{i,j}$, b^2_j .

3. Ejercicios

Para entrenar el regresor neuronal utilizaremos los mismos datos del TP1. Para poder evaluar la performance de nuestra red, dividiremos el dataset en dos conjuntos: uno que utilizaremos únicamente para el entrenamiento (315 muestras), y otro para la evaluación (99 muestras).

1. **Ejercicio**: Implementar el método *forward*() de la clase NeuralNetwork. Este método devuelve el resultado de evaluar la función f_θ. Las dimensiones de las matrices en juego están predefinidas e inicializadas en el código de base.

2. **Ejercicio**: Implementar el método *numerical_gradient*() que computa la derivada parcial de la función de perdida con respecto a cada uno de los parámetros de la red.

3. **Ejercicio**: Implementar el método *fit*() que realiza el ciclo de entrenamiento de la red. En cada iteración, se calcula el valor del gradiente promedio para todas las muestras del dataset y se actualizan los parámetros de la función utilizando esta dirección.

4. **Ejercicio**: Descargar el dataset, y entrenar la red. *Observación:* Es importante definir un criterio de convergencia. Graficar el valor de la función objetivo a lo largo del entrenamiento. Analizar brevemente cómo es su comportamiento.¿Cómo predice la red entrenada los datos de testing?, graficar el error cuadrático medio.

Notas y consejos: Entrenar una red neuronal es un proceso complejo debido a que hay muchos factores en juego, es por eso que la elección de los hiperparámetros es un punto clave. Podemos tener unos datos buenísimos y una arquitectura perfecta para el problema, pero si los hiperparámetros no están bien elegidos, difícilmente la red obtenga una buena solución. En nuestro caso, son la tasa de aprendizaje α, y también el ϵ con el que calculamos las derivadas parciales. Por este motivo probablemente van a tener que probar distintas configuraciones hasta que el entrenamiento de buenos resultados. Suele ser buena idea empezar con valores pequeños, del orden de 1×10^{-5}, e ir subiéndolos hasta ver cambios significativos en la función de pérdida. El entrenamiento es un proceso costoso computacionalmente y es probable que tarde en entrenar desde minutos a horas dependiendo del hardware que estén utilizando; por ese motivo, recomendamos trabajar ocn un subconjunto de muestras pequeño (10) en la etapa de implementación y dengueo.

4. Condiciones de entrega

Fecha final de entrega: 30 de junio - Modalidad de entrega: Vía campus

La entrega consiste en un archivo *TP2_apellido1_apellido2.ipynb*, con el código *Python* utilizado en formato jupyter-notebook (.ipynb). Agregar los gráficos y explicar utilizando markdown cada paso de su desarrollo. Justificar y analizar las decisiones tomadas a la hora de elegir valores para los hiperparámetros (α, ϵ, etc.). El trabajo debe realizarse en grupos de a dos (estricto). Pueden utilizar todas las bibliotecas que usamos en la materia (*numpy*, *matplotlib*, *pandas*). **No usar Pytorch, sklearn, tensorflow** o bibliotecas específicas de redes neuronales.