UNIVERSIDAD ADOLFO IBÁÑEZ

Tarea 1-TICS579

Deep Learning 2024-2

Profesor: Alfonso Tobar-Arancibia **Ayudante**: María Alejandra Bravo
Fecha de Entrega: 15/09/24 - 23:59 hrs. **Puntos Totales**: 12

INSTRUCCIONES:

- Entregue la tarea en un Jupyter Notebook nombrando el archivo con el siguiente formato: **Tarea_1_AT_MB.ipynb**. Donde en este caso, **AT** y **MB** corresponden a las iniciales de cada integrante del grupo.
- El notebook debe entregarse ejecutado con las celdas **en orden** y **sin errores de ejecución**.
- Recuerde que el código debe ser defendido en una sesión de defensa. **NO COPIE Y PEGUE CÓDIGO** que no entiende.
- No cumplir con las instrucciones implica nota 1.0.

Parte I: Optimizadores

1. Implemente utilizando Pytorch Autograd el **Update Rule** para minimizar la siguiente función:

$$f(\theta) = 0.5 \cdot \theta_0^2 + 12 \cdot \theta_1^2$$

donde se define el vector $\theta^T = [\theta_0, \theta_1]^T$. Considere $\alpha : 0.075$, $Epochs = 500$ y $\theta_{t=0}^T = [10, 1]$.

(a) (1 punto) Implemente el algoritmo de optimización **SGD** calculando derivadas utilizando Pytorch. Cree una lista para ir almacenando la evolución de θ_0 y θ_1 respectivamente. Grafique la evolución del parámetro con la función `plot_optimizers()` entregada en el notebook adjunto.

(b) (1 punto) Implemente el algoritmo de optimización **SGD con Momentum** siguiendo las mismas instrucciones que el paso anterior. En este caso considere además que $\beta = 0.9$

(c) (1 punto) Implemente el algoritmo de optimización **Adam** siguiendo las mismas instrucciones que los pasos anteriores. En este caso considere además que $\beta_1 = 0.9$ y $\beta_2 = 0.999$. Comente los resultados en comparación a los otros optimizadores.

Hint: Para extraer y modificar de variables a las que se les está calculando gradientes utilice `torch.tensor(...).data`.

Parte II: Red Softmax de dos Capas

2. Implemente una Red Softmax de dos capas según la definicion vista en clases utilizando Pytorch **(no puede usar Autograd acá)** para un problema de **clasificación multi-clase**. Para ello se utilizarán los siguientes datos:

   - X corresponde a una Matriz aleatoria de $m \times n$ con $m = 100$ y $n = 5$. Para garantizar resultados reproducibles utilice una **semilla aleatoria igual a 10** y genere los valores utilizando `torch.randn(...)`.
   - El vector de etiquetas y corresponderá a un vector aleatorio de 3 clases de dimensiones $m \times 1$.
   - La capa intermedia/oculta debe ser de 32 dimensiones para luego llegar a la capa de salida.
   - Utilice **CrossEntropy** como Loss Function, **ReLU** como función de Activación y $\alpha = 0.1$.

   (a) (1 punto) Defina una **función** `forward(...)` que reciba como parámetros X, y, W_1, W_2. Esta función debe retornar el valor de h_θ, S e I_y.

   (b) (1 punto) Defina una **función** que reciba una matriz y retorne la derivada de la función ReLU aplicada a dicha matriz.

   (c) (1 punto) Defina **funciones** que retornen los Gradientes de W_1 y W_2 (no está permitido utilizar `.backward()`). Cada función debe recibir sólo los parámetros necesarios para su cálculo.

   (d) (1 punto) Defina una **función** que represente el Entrenamiento de la Red.

   (e) (1 punto) Entrene la red y reporte el Accuracy para 10, 50, 100, 500 y 1000 epochs. Para ello utilice `accuracy_score(...)` de Scikit-Learn.

Parte III: Entrenando con Pytorch

3. Replique la parte II pero ahora utilizando Pytorch `nn.Module`.
   (a) (1 punto) Defina una clase que herede de `nn.Module` que defina la arquitectura de la Red. **Recuerde no usar bias**.
   (b) (1 punto) Define de manera apropiada el criterio y el optimizador para entrenar el modelo.
   (c) (1 punto) Defina el Training Loop utilizando todas las herramientas disponibles en Pytorch.
   (d) (1 punto) Entrene la red y reporte el Accuracy para 10, 50, 100, 500 y 1000 epochs. Para ello utilice `accuracy_score(...)` de Scikit-Learn. Compare sus resultados con los obtenidos en la Parte II.