Ejercicios1

June 6, 2024

1 Ejercicios primera semana

1.1 Ejercicios 1: Tensores

Crea los siguientes tensores utilizando PyTorch haciendo explícito el tipo de dato que contiene cada tensor:

1. $x^T = \begin{pmatrix} 1 & 3 & 7 & 8 & 11 & 345 & 1 \end{pmatrix}$
2. $x^T = \begin{pmatrix} 0.5, & 0.7 & 2 & 0.66 & 10 & 345 & 0.01 \end{pmatrix}$
3. $A = \begin{pmatrix} 10 & 0 & 15 \\ 2 & 9 & 12 \end{pmatrix}$
4. $B = \begin{pmatrix} 1 & 3 & 7 & 8 & 11 & 345 & 1 \\ 0.5 & 0.7 & 2 & 0.66 & 10 & 345 & 0.01 \end{pmatrix}$
5. T = [[[1, 7], [4, 10]], [[2, 8], [5, 11]], [[3, 9], [6, 12]]]
6. S = [[[4, 9], [4, 4]], [[8, 0], [1, 7]], [[2, 6], [5, 6]]]

Y a partir de estos realiza las siguientes operaciones:

1. Obtener el tamaño de cada tensor.
2. Obtener las sumas x + y, T + S
3. Obtener el producto externo entre x e y.
4. Obtener el producto de Haddamard entre T y S.
5. Obtener los productos Bx y By.
6. Existen dos forma de obtener un producto entre A y B usando transposición, encuentra estas formas y haz una multiplicaci´on con ambas matrices.

1.2 Ejercicio 2: Gráficas y manejo de tensores con PyTorch

Utilizando tensores en PyTorch con requerimiento de gradiente, realiza las gráficas computacionales en PyTorch de las siguientes funciones siguiendo las instrucciones en cada caso:

1. Define la función
$$f_1(x) = exp\{2x_1 + 2x_2 + 3x_3 + 0.5\}$$
utilizando 3 nodos (requiere tres variables).
2. Define la función
$$f_2(x) = \left(x_1^2 - 2x_2^2 + 4x_3^2\right)^2$$
utilizando 3 nodos (requiere tres variables).
3. Define la función
$$f_3(x) = \frac{\cos^2(x_1^2) + \cos^2(x_2^2) + \cos^2(x_3^2)}{||x||}$$

utilizando 5 nodos (requiere tres variables).

Para cada función utiliza una función de Python, siguiendo la siguiente estructura:

```
def f(x):
    node_one = #operación 1
    ...
    node_n = #operación n

    return #resultado
```

Genera un vector de $x = \begin{pmatrix} 1 & 3 & 4 \end{pmatrix}$ y obten su gradiente en cada caso por medio de las operaciones de backward. Por ejemplo, para la primera función se debe tener el resultado:

```
print(x.grad)
>>> tensor([1.5998e+09, 1.5998e+09, 2.3997e+09])
```

1.3 Ejercicio 3: Gráfica computacionales

Define los nodos de una gráfica computacional para una red neuronal generando los nodos como clases de python y utilizando únicamente numpy (no usar PyTorch, ni paqueterías especiales). Genera los nodos para las funciones siguientes:

1. Linear: $wx + b$ donde w y b se inician de forma aleatoria. Debe tomar como argumentos el tamaño de entrada del vector y el de salida.
2. Tanh: Función de activación de tangente hiperbólica definida como $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.
3. Softmax: Función de activación definida como $softmax(x_i) = \frac{e_i^x}{\sum_j e^{x_j}}$ donde cada x_i es una entrada del vector.
4. CrossEntropy: Función que toma dos argumentos x (vector) y y (un entero o clase) y que se estima de la forma $R(x) = -\sum_j \delta_{j,y} \ln x_j$ donde $\delta_{j,y} = 1$ si $j = y$ y 0 en otro caso.

Por ejemplo, la estructura del nodo para la función Linear es la siguiente:

```
class Linear():
    def __init__(self, input_size, output_size):
        self.grad = None
        #Inicializa w y b
        ...
    def __call__(self, x):
        # Realiza la función de wx+b
        ...
```

Un ejemplo del resultado es el siguiente:

```
l1 = Linear(2,3)
act1 = Tanh()
l2 = Linear(3,2)
out = Softmax()
criterion = CrossEntropy()

x = np.array([1,2])
```

```
pred = out(l2(act1(l1(x))))
loss = criterion(pred, 1)
print(pred)
print(loss)
>>>[0.31316141 0.68683859]
>>>0.3756559684082043
```