Ejercicios2

June 12, 2024

1 Ejercicios segunda semana

1.1 Ejercicio 1: Gráfica computacionales

Define los nodos de una gráfica computacional dinámica 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
```

1.2 Ejercicio 2: Cargadores de datos

Genera un cargador de datos a partir un dataset de tu preferencia. Este cargador de datos debe contrar con lo siguiente:

- El conjunto de datos x y las clases y, ambas en formato torch (recuerda que x debe ser flotante y y enteros largos).
- El tamaño del dataset size; es decir, el número total de datos que contiene.
- El tamaño de los tensores, sus dimensiones o número de features, feat_size. Si es necesario, realiza padding para que los datos sean del mismo tamaño.
- Una función de visualización plot que reduzca la dimensionalidad de los datos y los visualice en 2 dimensiones.

El resultado tiene que ser de la siguiente forma:

```
data = MyDataset('dataset_elegido')
print(data.size)
>> 10000
print(data.feat_size)
>> 16
data.plot()
>>
```

Crea además una clase de normalización por lotes que haga este tipo de normalización en los datos con base en un conjunto de parámetros a y b que deben aprenderese. Inicializa a como tensor de 1's y b como tensor de 0's. Recuerda utilizar torch.nn.Variable para declarar estos parámetros como parte del apendizaje:

```
a = torch.nn.Variables(torch.nn.ones(size))
```

Entonces aplica la normalización a cada lote y comprueba que la media y la desviación estándar son similares. Por ejemplo:

```
class BatchNormalization(torch.nn.Module):
    ### Aquí va la función para normalización por lotes
    ...

norm = BatchNormalization(data.feat_size, eps=1e-6)

for x,y in loader: ## loader es el cargador sobre los datos
    norm_x = norm(x)
    print(norm_x.mean(), norm_x.std())
```