Deep Learning with Pytorch

Carga de Datos. Uso de Dataset y DataLoader

Carga de datos

- La lectura de los datos (durante el entrenamiento) se hace por segmentos o batches. Estos se utilizan para el cálculo del error/coste y la actualización de los pesos.
- Siendo X e y dos tensores con las muestras y las etiquetas respectivamente. La lectura de los batches para entrenar sería de la siguiente forma:

```
for epoch in range(max_epochs):
    for i in n_batches:
        # Local batches and labels
        X_batch = X[i*batch_size:(i+1)*batch_size]
        y_batch = y[i*batch_size:(i+1)*batch_size]
        …
```

- Esta implementación hace difícil la realización de diversas operaciones sobre los datos (por ejemplo, leer los datos de forma no ordenada)

Usando Pytorch para la carga de datos

- Pytorch ofrece en `torch.utils.data` las funciones requeridas para crearnos nuestros propios dataset y cargadores de datos.
- Para crear un dataset que pueda ser utilizado por un cargador de datos DataLoader se extiende la clase `torch.utils.data.Dataset`. Se deben implementar (como mínimo) el constructor y dos funciones.
  - `__init__(self, samples, labels):` donde samples representaría la matriz X y labels y.
  - `__len__(self):` retorna el número de muestras del conjunto de datos.
  - `__getitem__(self, index):` retorna la muestra (y la etiqueta) en el índice index del conjunto de datos.
- U
- Una vez implementada esta clase, se podrán utilizar las funciones de `torch.utils.data.DataLoader` para la lectura de los datos.

Usando Pytorch para la carga de datos

```
from torch.utils.data import Dataset

class customDataset(Dataset):
    def __init__(self, X, y):
        self.X_data = torch.FloatTensor(X)
        self.y_data = torch.FloatTensor(y)
    def __len__(self):
        return len(self.X_data)
    def __getitem__(self, index):
        return self.X_data[index], self.y_data[index]

train_data = customDataset(X, y)
```

Usando Pytorch para la carga de datos

```
from torch.utils.data import Dataset, DataLoader
...
train_data = customDataset(X, y)
...
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,
shuffle=True)
...
for epoch in range(max_epochs):
    for X_batch, y_batch in train_loader:
        ...
```

- Dataloader tiene entre otros estos dos parámetros:
  - batch_size define el número de muestras del batch.
  - shuffle permite elegir si mezclar los datos a la hora de crear los batches (True). Así, que cada vez que se instancie Dataloader los datos se leerán de forma distinta.