Matemática Aplicada a Tecnologías de la Información
Curso 2023/24

Práctica 9: Graph Filters and Neural Networks (Parte 2)

Vamos a aprender cómo definir un model en `PyTorch`, así como a entrenarlo. Empleamos la práctica anterior para generar los datos.

```
N = 50 # number of nodes
S = sbm(n=N)
S = normalize_gso(S)
nTrain = 2000
nTest = 100
z = genera_difusion(grafo=S, n_muestras=nTrain+nTest)
x, y = data_from_diffusion(z)
trainData, testData = split_data(x, y, (nTrain,nTest)) #aqui usad
# la implementation que vosotros hayais usado para dividir
# en entrenamiento y test.
xTrain = trainData[0]
yTrain = trainData[1]
xTest = testData[0]
yTest = testData[1]

xTrain = torch.tensor(xTrain)
yTrain = torch.tensor(yTrain)

xTest = torch.tensor(xTest)
yTest = torch.tensor(yTest)
```

Ejercicio 1 (Partes de un entrenamiento) Al entrenar un modelo en Pytorch, debemos seguir los siguientes pasos.

1. Definir la arquitectura. En nuestro caso, un filtro usando la función de la práctica anterior. Por ejemplo:

   ```
   graphFilter = GraphFilter(S, 8)
   ```

2. Definir una función error y un optimizador. En nuestro caso, vamos a emplear:

   ```
   criterion = nn.MSELoss()
   optimizer = optim.Adam(graphFilter.parameters(), lr=learningRate)
   ```

3. Se crean listas vacías en la que almacenar los valores de la función error:

   ```
   train_loss_values = []
   test_loss_values = []
   epoch_count = []
   ```

4. Para cada época:

   4.1. Se pone el modelo en modo entrenamiento:

   ```
   graphFilter.train()
   ```

   4.2. Se evalúa el modelo para calcular la aproximación:

   ```
   y_pred = graphFilter(xTrain)
   ```

   4.3. Se calcula el error:

   ```
   loss = criterion(y_pred, yTrain)
   ```

   4.4. Se ponen los gradientes a cero:

   ```
   optimizer.zero_grad()
   ```

   4.5. Se retropropaga el error hacia atrás:

   ```
   loss.backward()
   ```

   4.6 Se actualiza el optimizador con los nuevos gradientes.

   ```
   optimizer.step()
   ```

5. En cada época, de manera opcional, podemos ir evaluando en el test. Así, podemos comprobar cómo va generalizando el modelo:

```
graphFilter.eval()
with torch.inference_mode():
  test_pred = graphFilter(xTest)
  test_loss = criterion(test_pred, yTest.type(torch.float))
  if epoch % 10 == 0:
        epoch_count.append(epoch)
        train_loss_values.append(loss.detach().numpy())
        test_loss_values.append(test_loss.detach().numpy())
        print(f"Epoch: {epoch} |
        MAE Train Loss: {loss} | MAE Test Loss: {test_loss} ")
```

6. Finalmente, representamos la evolución del error:

```
plt.plot(epoch_count, train_loss_values, label="Train loss")
plt.plot(epoch_count, test_loss_values, label="Test loss")
plt.title("Training and test loss curves")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend();
```

Ejercicio 2 (Emplear batches) Usando lo definido en el ejercicio anterior. Intenta redefinirlo empleando un parámetro llamado batches, de manera que en cada época se use solo un subconjunto aleatorio del conjunto de entrenamiento. ¿Mejora el rendimiento?