Práctica 3. Redes neuronales
Ejercicio a realizar

Alfons Juan, Jorge Civera

Departament de Sistemes
Informàtics i Computació

Índice

1. Construcción de una red neuronal

- ***Variante del tutorial oficial:*** cuaderno jupyter ***mlp***

https://github.com/apr-upv/apr2223/blob/master/mlp.ipynb

- Librerías:

```
import os
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
```

- Carga del conjunto de datos: ***FashionMNIST***

```
training_data = datasets.FashionMNIST(root="data", train=True,
    download=True, transform=ToTensor())
test_data = datasets.FashionMNIST(root="data", train=False,
    download=True, transform=ToTensor())
```

1.1. Preparación de los datos para entrenamiento

▷ ***DataLoader:*** procesa los datos definiendo batches de 64 muestras, barajando tras cada pasada completa por todos los datos

```
torch.manual_seed(23)
train_dataloader = DataLoader(training_data, batch_size=64,
  shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64,
  shuffle=True)
```

▷ El tamaño de batch ***batch_size*** es un parámetro que afecta al proceso de optimización

↦ ***batch_size*** ↑: gradientes más estables

↦ ***batch_size*** ↓: gradientes más inestables

1.2. Definición de la clase

- *Red:* subclase de **`nn.Module`**
- **`nn.ModuleList`**: lista de f. lineal seguida de f. de activación

```
class NeuralNetwork(nn.Module):
  def __init__(self, input_size, layers: list, num_classes):
    super(NeuralNetwork, self).__init__()
    self.flatten = nn.Flatten()
    self.layers = nn.ModuleList()
    self.input_size = input_size
    for output_size, activation_function in layers:
      self.layers.append(nn.Linear(input_size, output_size))
      input_size = output_size
      self.layers.append(activation_function)
    self.layers.append(nn.Linear(input_size, num_classes))
  def forward(self, x):
    x = self.flatten(x)
    for layer in self.layers:
      x = layer(x)
    return x
```

▷ *Instanciación:*

```
torch.manual_seed(23)
C = 10
N,H,W = training_data.data.shape; D = H*W
M1, M2 = 512, 512
model = NeuralNetwork(D, [(M1, nn.ReLU()), (M2, nn.ReLU())], C)
```

▷ *Parámetros de entrada de la clase `NeuralNetwork`:*

- ↦ `D`: dimensionalidad de los datos de entrada
- ↦ `[(M1, nn.ReLU()), (M2, nn.ReLU())]`: Lista de capas
- ↦ Una capa es una tupla (nº neuronas f. lineal, f. de activación)
- ↦ `C`: número de clases

1.3. Bucles de entrenamiento y test

```python
def train_loop(dataloader, model, loss_fn, optimizer):
  size = len(dataloader.dataset)
  for batch, (X, y) in enumerate(dataloader):
    pred = model(X); loss = loss_fn(pred, y)
    optimizer.zero_grad(); loss.backward(); optimizer.step() # backprop
    if batch % 100 == 0:
      loss, current = loss.item(), batch * len(X)
      print(f"trloss: {loss:>7f}  [{current:>5d}/{size:>5d}]")

def test_loop(dataloader, model, loss_fn):
  size = len(dataloader.dataset); nbatches = len(dataloader)
  teloss, correct = 0, 0
  with torch.no_grad():
    for X, y in dataloader:
      pred = model(X); teloss += loss_fn(pred, y).item()
      correct += (pred.argmax(1) == y).type(torch.float).sum().item()
  teloss /= nbatches; correct /= size
  print(f"teacc: {(100*correct):>0.1f}%, teloss: {teloss:>8f} \n")
```

1.4. Programa principal

▷ El optimizador ***Adam*** obtiene mejores resultados que ***SGD***

```
learning_rate = 1e-3
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader, model, loss_fn, optimizer)
    test_loop(test_dataloader, model, loss_fn)
print("Done!")
```

```
Epoch 1
-------------------------------
trloss: 2.29568 [    0/60000]
trloss: 0.82557 [ 6400/60000]
trloss: 0.36306 [12800/60000]
trloss: 0.40624 [19200/60000]
trloss: 0.47983 [25600/60000]
trloss: 0.42242 [32000/60000]
trloss: 0.25505 [38400/60000]
trloss: 0.33612 [44800/60000]
trloss: 0.47841 [51200/60000]
trloss: 0.32768 [57600/60000]
teacc: 84.5%, teloss: 0.41862

Epoch 2
-------------------------------
trloss: 0.27626 [    0/60000]
trloss: 0.37678 [ 6400/60000]
trloss: 0.22525 [12800/60000]
trloss: 0.32992 [19200/60000]
trloss: 0.42737 [25600/60000]
trloss: 0.11267 [32000/60000]
trloss: 0.27980 [38400/60000]
trloss: 0.29777 [44800/60000]
trloss: 0.31008 [51200/60000]
trloss: 0.27698 [57600/60000]
teacc: 85.7%, teloss: 0.39604

Epoch 3
-------------------------------
trloss: 0.39776 [    0/60000]
trloss: 0.26003 [ 6400/60000]
trloss: 0.39063 [12800/60000]
trloss: 0.45137 [19200/60000]
trloss: 0.37927 [25600/60000]
trloss: 0.42785 [32000/60000]
trloss: 0.35296 [38400/60000]
trloss: 0.25597 [44800/60000]
trloss: 0.37669 [51200/60000]
trloss: 0.47831 [57600/60000]
teacc: 87.7%, teloss: 0.33948

Epoch 4
-------------------------------
trloss: 0.30808 [    0/60000]
trloss: 0.18359 [ 6400/60000]
trloss: 0.26920 [12800/60000]
trloss: 0.13372 [19200/60000]
trloss: 0.40757 [25600/60000]
trloss: 0.27605 [32000/60000]
trloss: 0.25818 [38400/60000]
trloss: 0.15146 [44800/60000]
trloss: 0.25695 [51200/60000]
trloss: 0.21173 [57600/60000]
teacc: 86.3%, teloss: 0.36847

Epoch 5
-------------------------------
trloss: 0.34900 [    0/60000]
trloss: 0.27323 [ 6400/60000]
trloss: 0.15497 [12800/60000]
trloss: 0.17982 [19200/60000]
trloss: 0.20465 [25600/60000]
trloss: 0.19099 [32000/60000]
trloss: 0.23198 [38400/60000]
trloss: 0.26447 [44800/60000]
trloss: 0.18140 [51200/60000]
trloss: 0.28073 [57600/60000]
teacc: 87.8%, teloss: 0.34975

Epoch 6
-------------------------------
trloss: 0.21330 [    0/60000]
trloss: 0.25425 [ 6400/60000]
trloss: 0.23986 [12800/60000]
trloss: 0.18474 [19200/60000]
trloss: 0.29163 [25600/60000]
trloss: 0.21085 [32000/60000]
trloss: 0.24115 [38400/60000]
trloss: 0.28910 [44800/60000]
trloss: 0.19631 [51200/60000]
trloss: 0.26648 [57600/60000]
teacc: 88.1%, teloss: 0.33128

Epoch 7
-------------------------------
trloss: 0.23661 [    0/60000]
trloss: 0.40972 [ 6400/60000]
trloss: 0.23502 [12800/60000]
trloss: 0.23649 [19200/60000]
trloss: 0.42267 [25600/60000]
trloss: 0.13239 [32000/60000]
trloss: 0.29857 [38400/60000]
trloss: 0.15905 [44800/60000]
trloss: 0.20443 [51200/60000]
trloss: 0.31361 [57600/60000]
teacc: 88.6%, teloss: 0.33019

Epoch 8
-------------------------------
trloss: 0.41880 [    0/60000]
trloss: 0.21917 [ 6400/60000]
trloss: 0.29653 [12800/60000]
trloss: 0.22587 [19200/60000]
trloss: 0.27538 [25600/60000]
trloss: 0.32357 [32000/60000]
trloss: 0.15952 [38400/60000]
trloss: 0.20272 [44800/60000]
trloss: 0.23307 [51200/60000]
trloss: 0.16557 [57600/60000]
teacc: 88.3%, teloss: 0.32709

Epoch 9
-------------------------------
trloss: 0.24297 [    0/60000]
trloss: 0.20781 [ 6400/60000]
trloss: 0.22742 [12800/60000]
trloss: 0.43565 [19200/60000]
trloss: 0.41562 [25600/60000]
trloss: 0.28409 [32000/60000]
trloss: 0.14712 [38400/60000]
trloss: 0.30157 [44800/60000]
trloss: 0.20163 [51200/60000]
trloss: 0.24971 [57600/60000]
teacc: 88.8%, teloss: 0.31972

Epoch 10
-------------------------------
trloss: 0.15803 [    0/60000]
trloss: 0.14329 [ 6400/60000]
trloss: 0.15694 [12800/60000]
trloss: 0.22337 [19200/60000]
trloss: 0.26118 [25600/60000]
trloss: 0.29932 [32000/60000]
trloss: 0.36831 [38400/60000]
trloss: 0.22759 [44800/60000]
trloss: 0.22564 [51200/60000]
trloss: 0.29680 [57600/60000]
teacc: 88.7%, teloss: 0.33955

Done!
```

2. Ejercicio

- ▷ ***Objetivo:*** entrena un clasificador basado en redes neuronales que minimice el error de clasificación de ***FashionMNIST*** en test

- ▷ ***Principales parámetros ajustables:*** número de neuronas, número de épocas y factor de aprendizaje
  - ↦ Otros parámetros: nº de capas y tamaño de batch

- ▷ Entrega una memoria que describa los experimentos realizados para ajustar los parámetros del clasificador y el mejor resultado obtenido.