Clase22 IMA539

Mg. Alejandro Ferreira Vergara

June 19, 2023

1 Implementación de una CNN con Pytorch

[ ]:
```
import torch
# Modulo para computer vision
import torchvision
from torch import nn
from torchvision import transforms
from torch.utils.data import Subset
from torch.utils.data import DataLoader
import numpy as np
import matplotlib.pyplot as plt

# Carpeta root del dataset
image_path = 'dataset'

transform = transforms.Compose([transforms.ToTensor()])

mnist_dataset = torchvision.datasets.MNIST(root= image_path, train= True,
                                           transform= transform, download= True)

mnist_valid_dataset = Subset(mnist_dataset,
                             torch.arange(1000))

mnist_train_dataset = Subset(mnist_dataset,
                             torch.arange(1000, 11000))
                             #torch.arange(1000, len(mnist_dataset)))

mnist_test_dataset = torchvision.datasets.MNIST(root= image_path, train= False,
                                                transform= transform, download=␣
 ↪False)
```

[ ]:
```
fig = plt.figure(figsize=(12, 4))
for i in range(25):
    ax = fig.add_subplot(5, 5, i+1)
    ax.set_xticks([])
    ax.set_yticks([])
    img = mnist_train_dataset[i][0][0, :, :]
```

```
    ax.imshow(img, cmap= 'gray_r')
plt.show()
```

[ ]:
```
torch.manual_seed(1)
batch_size = 64

train_dl = DataLoader(mnist_train_dataset, batch_size= batch_size, shuffle=␣
 ↪True)
valid_dl = DataLoader(mnist_valid_dataset, batch_size= batch_size, shuffle=␣
 ↪False)
```

1.1 Modelo

[ ]:
```
model = nn.Sequential()

# Conv -> ReLU -> MaxPooling
model.add_module('conv1', nn.
 ↪Conv2d(in_channels=1,out_channels=32,kernel_size=5,padding=2))
model.add_module('relu1', nn.ReLU())
model.add_module('pool1', nn.MaxPool2d(kernel_size=2))

# Conv -> ReLU -> MaxPooling
model.add_module('conv2', nn.
 ↪Conv2d(in_channels=32,out_channels=64,kernel_size=5,padding=2))
model.add_module('relu2', nn.ReLU())
model.add_module('pool2', nn.MaxPool2d(kernel_size=2))

# Flatten
model.add_module('flatten', nn.Flatten())

# Full Connected -> ReLU -> Dropout
model.add_module('fc1', nn.Linear(3136,1024))
model.add_module('relu3', nn.ReLU())
model.add_module('dropout', nn.Dropout(p= .5))

# Full Connected
model.add_module('fc2', nn.Linear(1024, 10))
```

[ ]:
```
# Selección de la unidad de procesamiento
processing_unit = 'cuda' if torch.cuda.is_available() else 'cpu'
device = torch.device(processing_unit)

model.to(device)
print(f"You're using: {device} as device.")
```

Para ver un resumen del modelo, debemos intalar el siguiente módulo:

```
(ima539) ~ $ pip install torch-summary
```

[ ]:
```
import torchsummary

torchsummary.summary(model,
                     input_data= torch.randint(0, 255, (batch_size, 1, 28, 28))␣
 ↪/ 255,
                     col_names=["output_size", "num_params"], verbose= 0,␣
 ↪device= device)
```

[ ]:
```
loss_fn = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr= .001)
```

[ ]:
```
def train(model, num_epochs, train_dl, valid_dl, device):
    loss_hist_train = torch.zeros(num_epochs).to(device)
    accuracy_hist_train = torch.zeros(num_epochs).to(device)

    loss_hist_valid = torch.zeros(num_epochs).to(device)
    accuracy_hist_valid = torch.zeros(num_epochs).to(device)

    for epoch in range(num_epochs):
        model.train()
        for x_batch, y_batch in train_dl:
            x_batch, y_batch = x_batch.to(device), y_batch.to(device)
            pred = model(x_batch)
            loss = loss_fn(pred, y_batch)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

            loss_hist_train[epoch] += loss.item() * y_batch.size(0)
            is_correct = (torch.argmax(pred, dim=1) == y_batch).float()
            accuracy_hist_train[epoch] += is_correct.sum()

        loss_hist_train[epoch] /= len(train_dl.dataset)
        accuracy_hist_train[epoch] /= len(train_dl.dataset)

        model.eval()
        with torch.no_grad():
            for x_batch, y_batch in valid_dl:
                x_batch, y_batch = x_batch.to(device), y_batch.to(device)
                pred = model(x_batch)
                loss = loss_fn(pred, y_batch)
                loss_hist_valid[epoch] += loss.item() * y_batch.size(0)
                is_correct = (torch.argmax(pred, dim=1) == y_batch).float()
                accuracy_hist_valid[epoch] += is_correct.sum()
        loss_hist_valid[epoch] /= len(valid_dl.dataset)
        accuracy_hist_valid[epoch] /= len(valid_dl.dataset)
```

```
        print(f'Epoch {epoch+1} accuracy: {accuracy_hist_train[epoch]:.4f} '
              f'val_accuracy: {accuracy_hist_valid[epoch]:.4f}')

    return loss_hist_train.cpu(), loss_hist_valid.cpu(), accuracy_hist_train.
 ↪cpu(), accuracy_hist_valid.cpu()
```

[ ]:
```
torch.manual_seed(1)
num_epochs = 4
hist = train(model, num_epochs, train_dl, valid_dl, device)
```

[ ]:
```
x_arr = np.arange(len(hist[0])) + 1
fig = plt.figure(figsize= (12, 4))
ax = fig.add_subplot(1, 2, 1)
ax.plot(x_arr, hist[0], '-o', label='Train loss')
ax.plot(x_arr, hist[1], '--<', label='Validation loss')
ax.legend(fontsize=15)

ax = fig.add_subplot(1, 2, 2)
ax.plot(x_arr, hist[2], '-o', label='Train acc.')
ax.plot(x_arr, hist[3], '--<', label='Validation acc.')
ax.legend(fontsize=15)
ax.set_xlabel('Epoch', size=15)
ax.set_ylabel('Accuracy', size=15)
plt.show()
```

[ ]:
```
pred = model((mnist_test_dataset.data.unsqueeze(1) / 255).to(device))
is_correct = (torch.argmax(pred, dim=1) == mnist_test_dataset.targets.
 ↪to(device)).float()
print(f'Test accuracy: {is_correct.mean():.4f}')
```

[ ]:
```
fig = plt.figure(figsize=(12, 4))
for i in range(12):
    ax = fig.add_subplot(2, 6, i+1)
    ax.set_xticks([])
    ax.set_yticks([])
    img = mnist_test_dataset[i][0][0, :, :]
    pred = model(img.unsqueeze(0).unsqueeze(1).to(device))
    y_pred = torch.argmax(pred.cpu())
    ax.imshow(img, cmap= 'gray_r')
    ax.text(.9, .1, y_pred.item(), size= 15, color= 'blue',
            horizontalalignment= 'center', verticalalignment= 'center',
            transform= ax.transAxes)
plt.show()
```