Clase25 IMA539

Mg. Alejandro Ferreira Vergara

July 3, 2023

1 Redes Recurrentes con Pytorch

Para una mejor visualización del proceso de entrenamiento, podemos instalar:

```
pip install tqdm ipywidgets
```

[ ]:
```
import torch
import torchsummary
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
from tqdm.notebook import tqdm
```

[ ]:
```
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)
```

[ ]:
```
batch_size = 32
torch.manual_seed(1)
train_dl = DataLoader(mnist_train_dataset, batch_size= batch_size, shuffle=␣
 ↪True)
valid_dl = DataLoader(mnist_valid_dataset, batch_size= batch_size, shuffle=␣
 ↪False)
```

[ ]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

print(f'Esta implementación utilizará {device} para el entrenamiento e␣
 ↪inferencia del modelo')
```

[ ]:
```
sequence_length = 28
input_size = 28
hidden_size = 128
num_layers = 1
num_classes = 10
```

[ ]:
```
class CustomRNN(nn.Module):
    def __init__(self, input_size: int, hidden_size: int, num_layers: int,␣
 ↪num_classes: int, sequence_length: int):
        super(CustomRNN, self).__init__()
        # Hiperparámetros
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.num_classes = num_classes
        self.sequence_length = sequence_length

        # Capas
        self.rnn1 = nn.RNN(input_size, hidden_size, num_layers,␣
 ↪batch_first=True)
        self.rnn2 = nn.RNN(hidden_size, hidden_size, num_layers,␣
 ↪batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = x.reshape(-1, self.sequence_length, self.input_size)
        out, hidden = self.rnn1(x)
        out, hidden = self.rnn2(out)
        out = self.fc(out[:, -1, :])
        return out

model = CustomRNN(input_size, hidden_size, num_layers, num_classes,␣
 ↪sequence_length)
model.to(device)
```

[ ]:
```
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)
```

1.1 Entrenamiento

[ ]:
```
lr = 0.001
num_epochs = 15
```

[ ]:
```
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):
        with tqdm(train_dl, unit="batch") as tepoch:
            model.train()
            for x_batch, y_batch in tepoch:
                tepoch.set_description(f"Epoch {epoch} train")
                x_batch, y_batch = x_batch.to(device), y_batch.to(device)

                pred = model(x_batch)
                loss = loss_fn(pred, y_batch)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
                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()
                tepoch.set_postfix(loss= loss_hist_train[epoch].item() /␣
↪len(train_dl.dataset),
                                   accuracy= 100. * accuracy_hist_train[epoch].
↪item() / len(train_dl.dataset))

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

```
        with tqdm(valid_dl, unit="batch") as vepoch:
            model.eval()
            with torch.no_grad():
                for x_batch, y_batch in vepoch:
                    vepoch.set_description(f"Epoch {epoch} valid")
                    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()
                    vepoch.set_postfix(loss= loss_hist_valid[epoch].item() /␣
↪len(valid_dl.dataset),
                                       accuracy= 100. *␣
↪accuracy_hist_valid[epoch].item() / len(valid_dl.dataset))

            loss_hist_valid[epoch] /= len(valid_dl.dataset)
            accuracy_hist_valid[epoch] /= len(valid_dl.dataset)

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

[ ]:
```
torch.manual_seed(1)
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.set_xlabel('Epoch', size=15)
ax.set_ylabel('Loss', size=15)
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()
```

1.1.1 Testeo

[ ]:
```
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}')
```

[ ]:
```
path = 'rnn_model.pt'
torch.save(model, path)
```

[ ]:
```
import torch
import torchsummary
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
from tqdm.notebook import tqdm

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

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

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

print(f'Esta implementación utilizará {device} para el entrenamiento e␣
 ↪inferencia del modelo')

class CustomRNN(nn.Module):
    def __init__(self, input_size: int, hidden_size: int, num_layers: int,␣
 ↪num_classes: int, sequence_length: int):
        super(CustomRNN, self).__init__()
        # Hiperparámetros
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.num_classes = num_classes
        self.sequence_length = sequence_length

        # Capas
```

```
        self.rnn1 = nn.RNN(input_size, hidden_size, num_layers,␣
 ↪batch_first=True)
        self.rnn2 = nn.RNN(hidden_size, hidden_size, num_layers,␣
 ↪batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = x.reshape(-1, self.sequence_length, self.input_size)
        out, hidden = self.rnn1(x)
        out, hidden = self.rnn2(out)
        out = self.fc(out[:, -1, :])
        return out

path = 'rnn_model.pt'
model = torch.load(path)

model.eval()
```

[ ]:
```
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()
```

[ ]:
```
import torch
import torchsummary
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
from tqdm.notebook import tqdm

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)

batch_size = 32
torch.manual_seed(1)
train_dl = DataLoader(mnist_train_dataset, batch_size= batch_size, shuffle=␣
 ↪True)
valid_dl = DataLoader(mnist_valid_dataset, batch_size= batch_size, shuffle=␣
 ↪False)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

print(f'Esta implementación utilizará {device} para el entrenamiento e␣
 ↪inferencia del modelo')

sequence_length = 28
input_size = 28
hidden_size = 128
num_layers = 2
num_classes = 10
lr = 0.001
num_epochs = 5
```

[ ]:
```
class CustomRNN(nn.Module):
    def __init__(self, input_size: int, hidden_size: int, num_layers: int,␣
 ↪num_classes: int, sequence_length: int):
        super(CustomRNN, self).__init__()
        # Hiperparámetros
        self.input_size = input_size
```

```
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.num_classes = num_classes
        self.sequence_length = sequence_length

        # Capas
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,␣
 ↪batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        x = x.reshape(-1, self.sequence_length, self.input_size)
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).
 ↪to(device)
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).
 ↪to(device)
        out, hidden = self.lstm(x, (h0, c0))
        out = self.fc(out[:, -1, :])
        return out

model = CustomRNN(input_size, hidden_size, num_layers, num_classes,␣
 ↪sequence_length)
model.to(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):
        with tqdm(train_dl, unit="batch") as tepoch:
            model.train()
            for x_batch, y_batch in tepoch:
                tepoch.set_description(f"Epoch {epoch} train")
                x_batch, y_batch = x_batch.to(device), y_batch.to(device)

                pred = model(x_batch)
                loss = loss_fn(pred, y_batch)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
```

```
            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()
            tepoch.set_postfix(loss= loss_hist_train[epoch].item() /␣
↪len(train_dl.dataset),
                               accuracy= 100. * accuracy_hist_train[epoch].
↪item() / len(train_dl.dataset))

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

    with tqdm(valid_dl, unit="batch") as vepoch:
        model.eval()
        with torch.no_grad():
            for x_batch, y_batch in vepoch:
                vepoch.set_description(f"Epoch {epoch} valid")
                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()
                vepoch.set_postfix(loss= loss_hist_valid[epoch].item() /␣
↪len(valid_dl.dataset),
                                   accuracy= 100. *␣
↪accuracy_hist_valid[epoch].item() / len(valid_dl.dataset))

        loss_hist_valid[epoch] /= len(valid_dl.dataset)
        accuracy_hist_valid[epoch] /= len(valid_dl.dataset)

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

[ ]:
```
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)
```

[ ]:
```
torch.manual_seed(1)
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.set_xlabel('Epoch', size=15)
ax.set_ylabel('Loss', size=15)
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()
```