Lab 5 Concept-based Explainable AI

Teaching assistant: Gabriele Ciravegna (gabriele.ciravegna@polito.it) , Eleonora Poeta (eleonora.poeta@polito.it)

Lab 5: Concept-based XAI - CBM

CBM

Exercise 1

In this exercise you have to train a Concept Bottleneck Model (CBM) on the MNIST even/odd dataset.

Specifically you will:

1. Create the **MNIST even/odd Dataset**. In this variant of MNIST the task is to predict if the digit is even or odd.
    - In addition, you have to predict the **concepts** (in this case the **concepts are the digits**).
    - Create the train and test dataset.
2. Create the **CBM** model.
    - Instanciate the ResNet18 from torchvision.models
    - Retrieve the number of features from the Fully connect of the ResNet.
    - Create the digits (concepts) classifier. This is the concept bottleneck of the network.
    - Create the task classifier (even/odd).
3. Define the **loss** for the problem.
    - The CrossEntropy loss for the concepts.
    - The CrossEntropy loss for the task.
4. Train and test the model.
5. Visualize a few predictions.

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
```

Exercise on Concept Bottleneck Models (CBM)

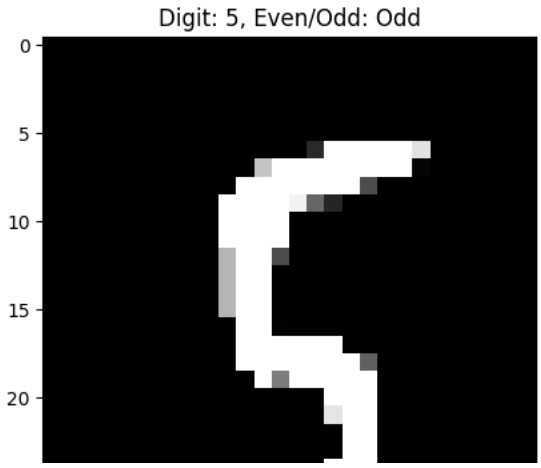
In this part of the laboratory you will have to train a CBM.

We will do it on a toy dataset MNIST Even-Odd. This dataset is a variant of MNIST where you not only have to predict the digit but also if it is even or odd.

You can create the dataset using the following code:

```
# Class to create the MNIST-EVENODD dataset
class MNISTEvenOdd(torchvision.datasets.MNIST):

    # Constructor for the MNISTEvenOdd class
    def __init__(self, root, train=True, download=True):
        transform = transforms.Compose([transforms.ToTensor(),
                                        transforms.Normalize((0.1307,), (0.3081,)), # MNIST mean and std
                                        transforms.Lambda(lambda x: x.repeat(3, 1, 1)) # Repeat the image in 3 channels
                                        ])

        # Call the constructor of the parent class (MNIST)
        super(MNISTEvenOdd, self).__init__(root, train, transform, download=download)

        # Reduce the size of the dataset for the exercise
        self.data = self.data[:10000]
        self.targets = self.targets[:10000]

    # Method to get an item from the dataset
    def __getitem__(self, index):
        # Get the image and target at the given index from the parent class
        img, target = super(MNISTEvenOdd, self).__getitem__(index)

        # define the concept label
        concept = target

        # define the task label as even or odd
        task = target % 2

        # Return the image, concept and task labels
        return img, concept, task

# instantiate the dataset
train_dataset = MNISTEvenOdd(root='.',
                             train=True)

test_dataset = MNISTEvenOdd(root='.',
                            train=False)

# Visualize the dataset
import matplotlib.pyplot as plt

# Get the first image and target from the training dataset
image, target, even_odd = train_dataset[100]

# Plot the image
plt.imshow(image.squeeze().numpy().transpose(1,2,0), cmap='gray')
plt.title(f'Digit: {target}, Even/Odd: {"Odd" if even_odd == 1 else "Even"}')
plt.show()
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ul
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ul
100%|██████████| 9912422/9912422 [00:00<00:00, 33378863.69it/s]
Extracting ./MNISTEvenOdd/raw/train-images-idx3-ubyte.gz to ./MNISTEvenOdd/ra

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ul
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ul
100%|██████████| 28881/28881 [00:00<00:00, 1120889.91it/s]Extracting ./MNISTE

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz

Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ub
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ub
100%|██████████| 1648877/1648877 [00:00<00:00, 8799079.11it/s]
Extracting ./MNISTEvenOdd/raw/t10k-images-idx3-ubyte.gz to ./MNISTEvenOdd/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 403: Forbidden

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ub
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ub
100%|██████████| 4542/4542 [00:00<00:00, 6589598.33it/s]Extracting ./MNISTEve


WARNING:matplotlib.image:Clipping input data to the valid range for imshow wi
```



```
# Create the CBM model.

#It should have a ResNet18 as the backbone and two classifying layers: one for the digit and one on top for the even/odd cla

class CBM(nn.Module):
    def __init__(self, num_concepts=10, num_classes=2):
        super(CBM, self).__init__()
        # Load the ResNet18 model
        self.resnet = torchvision.models.resnet18(pretrained=False)

        # Get the number of features in the ResNet18 classifier
        num_ftrs = self.resnet.fc.in_features

        self.resnet.fc = nn.Identity()

        # Create the digit classifier
        self.digit_classifier = nn.Linear(num_ftrs, num_concepts)

        # Create the even/odd classifier on top of the digit classifier
        self.even_odd_classifier = nn.Linear(num_concepts, num_classes)

    def forward(self, x):
        # Forward pass through the ResNet18
        x = self.resnet(x)

        # Forward pass through the digit classifier
        digit = self.digit_classifier(x)

        # Forward pass through the even/odd classifier
        even_odd = self.even_odd_classifier(digit)

        return digit, even_odd

# Create the model
model = CBM()
model = model.eval()

# Make a forward pass with the model
digit, even_odd = model(image.unsqueeze(0))

# Print the output shapes and the associated logits with two digits precision
print('Digit:', digit.shape, digit.detach().numpy().round(2))
print('Even/Odd:', even_odd.shape, even_odd.detach().numpy().round(2))
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is dep
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enu
  warnings.warn(msg)
Digit: torch.Size([1, 10]) [[ 0.28  0.19 -0.21 -0.19  0.04  0.22 -0.28  0.12 -0.34 -0.38]]
Even/Odd: torch.Size([1, 2]) [[0.22 0.03]]
```

```
# train the model
# train the model
import torch.optim as optim
from torch.utils.data import DataLoader

# Define the batch size
batch_size = 128

# Define the number of epochs
num_epochs = 2

# Create the data loaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

# Define the loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)

device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Training on {device}")
```

```
Training on cpu
```

```
# Train the model
# Iterate over epochs
model = model.to(device)

for epoch in range(num_epochs):

    # Set the model in training mode
    model.train()

    # Iterate over the batches
    for i, (images, concepts, tasks) in enumerate(train_loader):
        # put the image and labels on the device
        images, concepts, tasks = images.to(device), concepts.to(device), tasks.to(device)

        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass
        digit, even_odd = model(images)

        # Compute the loss for the digit classification
        loss_digit = criterion(digit, concepts)

        # Compute the loss for the even/odd classification
        loss_even_odd = criterion(even_odd, tasks)

        # Parameter gamma
        gamma = 0.7

        # Compute the total loss
        loss = loss_digit + gamma * loss_even_odd

        # Backward pass
        loss.backward()

        # Optimize
        optimizer.step()

        # Print the loss every 10 iterations
        if (i+1) % 10 == 0:
            print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs, i+1, len(train_loader), loss.item(

    # Evaluate the model on the test set
    model.eval()

    # Initialize the accuracy
    concept_acc = 0
    task_acc = 0

    # Disable gradient computation
    with torch.no_grad():

        # Iterate over the test set
        for images, concepts, tasks in test_loader:
            # put the image and labels on the device
            images, concepts, tasks = images.to(device), concepts.to(device), tasks.to(device)

            # Forward pass
            digit, even_odd = model(images)

            # Get the predicted digit
            _, predicted_digit = torch.max(digit.data, 1)

            # Get the predicted even/odd
            _, predicted_even_odd = torch.max(even_odd.data, 1)

            # Update the number of correct predictions
            concept_acc += (predicted_digit == concepts).sum().item()
            task_acc += (predicted_even_odd == tasks).sum().item()

    # Compute the accuracy
    concept_acc = concept_acc / len(test_dataset)
    task_acc = task_acc / len(test_dataset)

    # Print the accuracy
    print(f'Epoch [{epoch+1}/{num_epochs}], Concept Accuracy: {concept_acc:.2f}, Task Accuracy: {task_acc:.2f}')


    # Visualize a few predictions
    import numpy as np
```

```
Epoch [1/2], Step [10/79], Loss: 0.4035
Epoch [1/2], Step [20/79], Loss: 0.2392
Epoch [1/2], Step [30/79], Loss: 0.2278
Epoch [1/2], Step [40/79], Loss: 0.1377
Epoch [1/2], Step [50/79], Loss: 0.3047
Epoch [1/2], Step [60/79], Loss: 0.3425
Epoch [1/2], Step [70/79], Loss: 0.0820
Epoch [1/2], Concept Accuracy: 0.97, Task Accuracy: 0.98
```