Evaluation UmmonHealthTech

March 15, 2024

0.1 Test Technique

L'objectif de ce travail est de transformer le jeu de données MNIST, composé d'images de chiffres manuscrits, en un problème d'apprentissage par instances multiples (MIL). Cette transformation implique de regrouper les images en sacs d'instances et d'attribuer des étiquettes de sac en fonction de la présence de certaines caractéristiques dans les instances individuelles. Ensuite, un modèle de classification sera mis en place pour prédire la présence de ces caractéristiques dans les sacs d'instances. L'évaluation de ce modèle sera réalisée en calculant sa précision sur un ensemble de données de test. Cette approche vise à explorer la capacité du modèle à généraliser à partir de données d'entraînement complexes et à prendre des décisions précises sur de nouveaux ensembles de données.

[2]:
```
# Importation des bibliothèques et modules nécessaires
import torch
from torchvision import datasets, transforms
import torch.nn as nn
import torch.optim as optim
import numpy as np
```

[3]:
```
# Téléchargement du jeu de données MNIST
mnist_train = datasets.MNIST(root='./data', train=True, download=True,␣
 ↪transform=transforms.ToTensor())
mnist_test = datasets.MNIST(root='./data', train=False, download=True,␣
 ↪transform=transforms.ToTensor())
```

[4]:
```
# Fonction de création d'un ensemble de données mil-mnist et de regroupement␣
 ↪des images mnist en sacs
def mil_mnist(donnees):
    sacs = []
    etiquettes_sac = []

    for i in range(len(donnees)):
        taille_sac = torch.randint(low=5, high=10, size=(1,)).item()
        indices_sac = torch.randint(low=0, high=len(donnees),␣
 ↪size=(taille_sac,))
        sac = torch.stack([donnees[idx][0] for idx in indices_sac])
        sacs.append(sac)
```

```
        etiquettes = [donnees[idx][1] for idx in indices_sac]
        etiquette_sac = 1 if 1 in etiquettes else 0
        etiquettes_sac.append(etiquette_sac)

    return sacs, etiquettes_sac
```

[5]:
```
# Création d'une collection de données MIL-MNIST
mil_train_data, mil_train_etiq = mil_mnist(mnist_train)
mil_test_data, mil_test_etiq = mil_mnist(mnist_test)
```

[6]:
```
def gener_donnees(target_number=1, bag_length=64, num_train_bags=100,␣
 ↪num_test_bags=50):
    train_data = []
    train_etiq = []
    test_data = []
    test_etiq = []

    for _ in range(num_train_bags):
        sac = np.random.randint(0, 10, size=bag_length)
        train_data.append(sac)
        train_etiq.append(1 if target_number in sac else 0)

    for _ in range(num_test_bags):
        sac = np.random.randint(0, 10, size=bag_length)
        test_data.append(sac)
        test_etiq.append(1 if target_number in sac else 0)

    return np.array(train_data), np.array(train_etiq), np.array(test_data), np.
 ↪array(test_etiq)
```

[7]:
```
train_data, train_etiq, test_data, test_etiq = gener_donnees(target_number=1,␣
 ↪bag_length=64, num_train_bags=100, num_test_bags=50)
```

[8]:
```
# convertion de chaque image en vecteur.
train_data = torch.stack([torch.tensor(sac).float().view(-1) for sac in␣
 ↪train_data])
test_data = torch.stack([torch.tensor(sac).float().view(-1) for sac in␣
 ↪test_data])
train_etiq = torch.tensor(train_etiq).float()
test_etiq = torch.tensor(test_etiq).float()
```

[9]:
```
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(64, 128)
        self.fc2 = nn.Linear(128, 64)
```

```
        self.fc3 = nn.Linear(64, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x
```

[10]:
```
# Initialisation du model SimpleNN
model = SimpleNN()
```

[11]:
```
perte = nn.BCELoss() # Fontion de perte
optimizer = optim.Adam(model.parameters(), lr=0.001) # Optimiseur
```

[12]:
```
# Entraînement
num_epochs = 100
for epoch in range(num_epochs):
    optimizer.zero_grad()
    sort = model(train_data)
    loss = perte(sort, train_etiq.view(-1, 1))
    loss.backward()
    optimizer.step()
```

[13]:
```
# Evaluation du modèle
with torch.no_grad():
    test_sort = model(test_data)
    test_predictions = (test_sort > 0.5).float()
    precision = (test_predictions == torch.FloatTensor(test_etiq).view(-1, 1)).
 ↪float().mean()
```

[14]:
```
print("Precision sur nos données test:", precision.item())
```

```
Precision sur nos données test: 1.0
```

0.2 Conclusion

Le code a été correctement mis en œuvre pour transformer le jeu de données MNIST en un problème d'apprentissage par instances multiples (MIL) et pour évaluer un modèle de classification simple. La précision obtenue sur les données de test indique la capacité du modèle à bien généraliser pour cette tâche spécifique.

[ ]:
```
```