主題描述：

這是一個基於深度學習的神經網絡模型的實現，使用PyTorch的機器學習框架，並結合Ludwig中的一些模塊。模型通過編碼器將輸入數據轉換為有用的表示形式，然後利用組合器將不同編碼器的輸出整合成一個統一的表示，最後由解碼器根據組合器的輸出生成最終的預測結果。另外此模型包含了損失計算和性能評估的功能，其中損失函數計算模型預測與真實值之間的差異，並在訓練過程中最小化這差異，而性能評估指標則用於衡量模型在特定任務上的表現。

引用資料與程式碼敘述：

1、SimpleModel：這是模型的類名，代表這個程式的主要功能定義為一個簡單的神經網絡模型。模型的架構相對較簡單，只包含兩個全連接層，並且適用於處理MNIST數據集的分類任務。

程式碼：

# 定義模型／模型架構修改

class SimpleModel(nn.Module):

def \_\_init\_\_(self):

super(SimpleModel, self).\_\_init\_\_()

self.fc1 = nn.Linear(784, 128)

self.fc2 = nn.Linear(128, 10)

def forward(self, x):

x = torch.flatten(x, start\_dim=1)

x = self.fc1(x)

x = torch.relu(x)

x = self.fc2(x)

return x

2、MNIST：這個程式主要是用來處理 MNIST 數據集的，包括下載和加載數據、定義模型、訓練和測試模型等。因此，將「MNIST」包含在程式名稱中可以清楚地表明這個程式的用途。

程式碼：

import torchvision

import torchvision.transforms as transforms

# 下載和加載MNIST數據集

transform = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.5,), (0.5,))

])

trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch\_size=64, shuffle=True)

testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch\_size=64, shuffle=False)

3、Classifier：由於這個程式是用來建立一個分類器模型，它用於分類手寫數字圖像。

自行部份修改對比：

1. 模組結構

原始程式：

* 使用Ludwig庫中的LudwigModule作為基類。
* 定義input\_features和output\_features作為編碼器和解碼器，分別從 ENCODER\_REGISTRY和DECODER\_REGISTRY中實例化。
* 定義一個combiner模組來組合編碼器的輸出。

修改後的程式：

* 不再使用Ludwig庫，改用純PyTorch的nn.Module作為基類。
* 定義簡單的Encoder和Decoder類，分別使用GRU層。
* 去掉combiner，直接將編碼器的輸出傳遞給解碼器。

1. 前向傳播

原始程式：在forward方法中，先通過各個編碼器對輸入進行編碼，然後將編碼結果傳遞給combiner，最後將combiner的輸出傳遞給各個解碼器，得到最終輸出。

修改後的程式：在forward方法中，輸入直接傳遞給編碼器，然後將編碼器的輸出應用dropout後傳遞給解碼器，得到最終輸出。

1. 正則化

原始程式：沒有明確的正則化策略。

修改後的程式：

* 添加dropout層來進行正則化。
* 在get\_loss方法中增加了L2正則化項。

1. 損失計算

原始程式：get\_loss方法中，通過遍歷各個解碼器來計算總損失。

修改後的程式：get\_loss方法計算所有模型參數的L2損失，並返回。

心得：

修改後的程式在簡化架構並增強針對性方面取得了顯著進展。通過引入dropout和L2正規化，提高了模型的泛化能力和性能。不依賴於Ludwig庫的設計使得程式碼更加清晰和易於維護，這對於開發特定任務的機器學習模型至關重要。整體而言，這些調整使得我們能夠更靈活地應對不同的需求，同時保持程式碼的簡潔性和可讀性。