REPORT MINOR 01

PRINCY GAUTAM
(B20BB051)

Question 01

The code is a script for training a convolutional neural network (CNN) for image classification. The script performs the following steps:

1. Load the MNIST dataset from the 'openml' repository and filter the data to include only the target 5 classes (1, 3, 5, 7, 9).
2. Split the filtered data into training and test sets using the 'train_test_split' function from scikit-learn.
3. Convert the data into tensors and normalize the values to range between 0 and 1.
4. Define a custom dataset class called MNIST_Dataset, which transforms the data and labels into a format that is compatible with PyTorch's DataLoader.
5. Apply random horizontal flip and gaussian noise on the training data using the RandomGaussianNoise and Compose from torchvision.transforms.
6. Create the custom datasets and 'data loaders' using the MNIST_Dataset class and DataLoader class from PyTorch.
7. Define a weight initialization function, which sets the initial values of the weights in the convolutional layers using the xavier_normal_ method from PyTorch.
8. Define the CNN model class, which has three convolutional layers, two pooling layers, and one fully connected layer.
9. Train the CNN model using an Adam optimizer and a cross-entropy loss function.
10. Evaluate the performance of the trained model on the test set.

Question 02

This code is a PyTorch implementation of an autoencoder for the MNIST dataset, with a restriction that only the digits with labels 1, 3, 5, 7, and 9 are used.

The autoencoder consists of two parts: an encoder and a decoder. The encoder consists of three 2D convolutional layers, each followed by a ReLU activation function.

The decoder consists of three transposed 2D convolutional layers, again each followed by a ReLU activation function.

The autoencoder is trained for 10 epochs on a subset of the MNIST training set that only contains the digits 1, 3, 5, 7, and 9. The Adam optimizer is used with a learning rate of 0.001.

The mean squared error (MSE) loss function is used to measure the difference between the output of the autoencoder and the original images.

Finally, the trained autoencoder is evaluated on a subset of the MNIST test set that only contains the digits 1, 3, 5, 7, and 9. The evaluation of the model is based on the mean squared error (MSE) loss.

Comparisons

In CNN model as follows:

```
CNN(
  (conv1): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool1): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (conv2): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
  (pool2): AvgPool2d(kernel_size=2, stride=2, padding=0)
  (conv3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
  (fc1): Linear(in_features=1568, out_features=512, bias=True)
)
```

The loss observed is as follows:

```
Epoch 1 loss: 0.377
Epoch 2 loss: 0.120
Epoch 3 loss: 0.088
Epoch 4 loss: 0.069
Epoch 5 loss: 0.061
Epoch 6 loss: 0.052
Epoch 7 loss: 0.047
Epoch 8 loss: 0.041
Epoch 9 loss: 0.039
Epoch 10 loss: 0.036
Finished training
```

The test accuracy is as follows:

```
Accuracy of the network on the test images: 98 %
```

In autoencoder model as follows:

```
Autoencoder(
  (encoder): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(8, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (3): ReLU()
    (4): Conv2d(16, 32, kernel_size=(7, 7), stride=(1, 1))
  )
  (decoder): Sequential(
    (0): ConvTranspose2d(32, 16, kernel_size=(7, 7), stride=(1, 1))
    (1): ReLU()
    (2): ConvTranspose2d(16, 8, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), output_padding=(1, 1))
    (3): ReLU()
    (4): ConvTranspose2d(8, 1, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), output_padding=(1, 1))
  )
)
```

The loss in observed as:

```
Epoch [1/10], Step [954/954], Loss: 0.0623
Epoch [2/10], Step [954/954], Loss: 0.0518
Epoch [3/10], Step [954/954], Loss: 0.0338
Epoch [4/10], Step [954/954], Loss: 0.0335
Epoch [5/10], Step [954/954], Loss: 0.0312
Epoch [6/10], Step [954/954], Loss: 0.0276
Epoch [7/10], Step [954/954], Loss: 0.0332
Epoch [8/10], Step [954/954], Loss: 0.0314
Epoch [9/10], Step [954/954], Loss: 0.0297
Epoch [10/10], Step [954/954], Loss: 0.0261
```

The decoder is dropped and 1 FC layer with 512 nodes are stacked and the loss now observed is as follows:

```
Epoch 1 loss: 0.359
Epoch 2 loss: 0.058
Epoch 3 loss: 0.036
Epoch 4 loss: 0.027
Epoch 5 loss: 0.023
Epoch 6 loss: 0.019
Epoch 7 loss: 0.016
Epoch 8 loss: 0.014
Epoch 9 loss: 0.014
Epoch 10 loss: 0.012
Finished Training
```

The test accuracy is observed as follows:

```
Test accuracy: 99 %
```

Both the models have approximately the same accuracy on the test images. However, the autoencoder model has higher accuracy than the CNN model by 1%.

Hyperparameters

The hyperparameters used in the Question 02 are:

1. **Batch size: 32**

    - A batch size of 32 is a common choice and can provide a good starting point for experimentation. However, larger batch sizes can also require more memory and may cause the model to overfit to the training data, leading to reduced generalization performance.

2. **Learning rate (lr): 0.001**

    - The learning rate is a hyperparameter that determines how quickly the model learns.

    - If the learning rate is set too high, the model may overshoot the optimal solution and never converge. If the learning rate is set too low, the model may converge slowly or not at all.

    - A smaller learning rate, such as 0.001, allows the model to converge more slowly and steadily, while a larger learning rate, such as 0.01 or 0.1, may lead to faster convergence, but with a higher risk of oscillating around the optimal solution.

    - Ultimately, the best learning rate for a particular problem depends on many factors, including the architecture of the model, the size of the dataset, and the complexity of the task. The value of 0.001 is a good starting point, but it was adjusted based on the results of experimentation and tuning.

3. **Number of epochs: 10**

    - The number of epochs is a hyperparameter that determines the number of times the model will be trained on the entire training dataset.

    - In the above code, the number of epochs is set to 10, which means that the model will be trained on the training dataset 10 times.

- If the number of epochs is too small, the model may not have enough time to learn from the training data, and if it is too high, the model may start overfitting to the training data.

4. **Optimizer: Adam**

   - The Adam optimizer is a popular choice in deep learning because it is computationally efficient and can provide good results with little hyperparameter tuning. The Adam algorithm uses the idea of adaptive learning rates, which means that different parameters are updated at different rates. This helps the optimizer converge faster and with better accuracy.

   - The use of the Adam optimizer in the code is a hyperparameter choice that is adjusted based on the specific problem and desired outcome.

5. **Loss function: Mean Squared Error Loss (MSELoss)**

   - The Mean Squared Error Loss (MSELoss) is used in the code as the loss function because it measures the average squared differences between the predicted and actual output values. It's commonly used in regression problems, where the goal is to predict a continuous output value, rather than a class label.

6. **Loss function: Negative log - likelihood loss (NLLLoss)**

   - In the above code, the choice of NLLLoss as the loss function is a hyperparameter. It represents a hyperparameter because it can be set or chosen by the practitioner, and the value can impact the training and performance of the model. For example, using a mean squared error loss instead of a different loss function, such as binary cross-entropy, can affect the training behavior and final performance of the model.

References

- https://www.geeksforgeeks.org/applying-convolutional-neural-network-on-mnist-dataset/
- https://www.geeksforgeeks.org/implementing-an-autoencoder-in-pytorch/
- https://medium.com/dataseries/convolutional-autoencoder-in-pytorch-on-mnist-dataset-d65145c132ac