Academic Reflective Journal structure

This reflection delves into my experience with a deep learning lab focused on CNNs, emphasizing the roles of various layers, encoding methods, and the optimization process. It aims to highlight my learning journey, particularly the insights gained from overcoming a coding error.

The lab's core revolved around constructing a CNN for digit classification, introducing me to Conv2D layers for feature extraction, the theoretical purpose of MaxPooling2D layers for dimensionality reduction, and the practical necessity of one-hot encoding for categorical data processing. The Flatten layer's role in transitioning to fully connected layers was pivotal. The choice of optimizer and loss function, crucial for model training, was a significant learning aspect.

Initially, the complexity of CNN architecture was overwhelming. The coding error, which took considerable effort to diagnose and resolve, was particularly challenging but turned into a profound learning opportunity. It underscored the importance of meticulous attention to detail in deep learning projects.

Navigating through the error and its resolution process enriched my understanding of model-device compatibility in PyTorch. It highlighted the nuanced aspects of deploying neural networks, enhancing my analytical skills in troubleshooting and optimization.

The lab bridged theoretical concepts with practical application, reinforcing the importance of data preprocessing, the mechanics of convolutional and pooling layers, and the dynamics of backpropagation in model training.

Critically reflecting on the error and its resolution led to a deeper appreciation of the underlying principles of CNNs and the PyTorch framework. It prompted considerations on alternative approaches and the importance of cross-validating model configurations with hardware capabilities.

This experience fostered personal growth in problem-solving and resilience. It developed my technical skills in deep learning, offering insights into effective debugging and model fine-tuning practices.

The lessons learned, especially from resolving the coding error, will be invaluable in future deep learning projects. They've equipped me with a robust foundation for exploring more complex models and applications.

Reflecting on this lab has been immensely rewarding, offering a holistic view of learning and growth through practical challenges. The journey from confronting a coding error to achieving a successful model implementation has been enlightening, emphasizing the iterative and exploratory nature of working with deep learning technologies.

```
input_size = 26 * 26 * 32  # Flattened dimension for the linear layer

############### CODE HERE ###############
import torch.nn.functional as F
# Define the CNN model class
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # Convolution layer: 1 input channel, 32 output channels, kernel size 3
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3, stride=1)
        # Calculate output size after convolution to use in linear layer
        self.output_size_after_conv = (28 - 3) // 1 + 1  # For 28x28 input images
        # First fully connected layer: flattened size to 128 nodes
        self.fc1 = nn.Linear(32 * self.output_size_after_conv ** 2, 128)
        # Output layer: 128 input features to 10 output classes (digits 0-9)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        # Apply ReLU activation function after convolution
        x = F.relu(self.conv1(x))
        # Flatten the output of conv layers to feed into the linear layer
        x = x.view(-1, 32 * self.output_size_after_conv ** 2)
        # Apply ReLU activation function after first fully connected layer
        x = F.relu(self.fc1(x))
        # No activation function before softmax; PyTorch combines LogSoftmax and NLLLoss
        x = self.fc2(x)
        # Return log probability; use LogSoftmax for numerical stability
        return F.log_softmax(x, dim=1)

# Create an instance of the CNN
net = CNN()
net = net.to(device)
############### END OF CODE ###############

def xavier_init_weights(m):
    if type(m) == nn.Linear:
        torch.nn.init.xavier_uniform_(m.weight)

# Initialize weights/parameters for the network
net.apply(xavier_init_weights)
```

Training the network

Now you are ready to train the CNN.

```
import time

# Network training and validation

# Start the outer epoch loop (epoch = full pass through the dataset)
for epoch in range(num_epochs):
    start = time.time()

    training_loss = 0.0

    # Training loop (with autograd and trainer steps)
    # This loop trains the neural network
    # Weights are updated here
    net.train()  # Activate training mode (dropouts and so on)
    for images, target in train_loader:
        # Zero the parameter gradients
        optimizer.zero_grad()
        images = images.to(device)
        target = target.to(device)
        # Forward + backward + optimize
        output = net(images)
        L = loss(output, target)
        L.backward()
        optimizer.step()
        # Add batch loss
        training_loss += L.item()

    # Take the average losses
    training_loss = training_loss / len(train_loader)

    end = time.time()
    print("Epoch %s. Train_loss %f Seconds %f" % (epoch, training_loss, end - start))
```

```
Epoch 0. Train_loss 0.255611 Seconds 25.233345
Epoch 1. Train_loss 0.074579 Seconds 6.989597
Epoch 2. Train_loss 0.044465 Seconds 6.990955
Epoch 3. Train_loss 0.030154 Seconds 6.997209
Epoch 4. Train_loss 0.020868 Seconds 7.002403
Epoch 5. Train_loss 0.014300 Seconds 6.998096
Epoch 6. Train_loss 0.009187 Seconds 7.004603
Epoch 7. Train_loss 0.006953 Seconds 6.989209
Epoch 8. Train_loss 0.006399 Seconds 6.980443
Epoch 9. Train_loss 0.005275 Seconds 6.980605
```

Testing the network

Finally, evaluate the performance of the trained network on the test set.

```
from sklearn.metrics import classification_report

net.eval()  # Activate eval mode (don't use dropouts and such)

# Get test predictions
predictions, labels = [], []
for images, target in test_loader:
    images = images.to(device)
    target = target.to(device)

    predictions.extend(net(images).argmax(axis=1).tolist())
    labels.extend(target.tolist())

# Print performance on the test data
print(classification_report(labels, predictions, zero_division=1))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.97      0.99      0.98      1032
           3       0.98      0.98      0.98      1010
           4       0.99      0.98      0.98       982
           5       0.98      0.98      0.98       892
           6       0.99      0.97      0.98       958
           7       0.99      0.98      0.98      1028
           8       0.98      0.97      0.98       974
           9       0.97      0.98      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```