2.2 Neural Network

1. In this PyTorch basics and back-propagation section, you performed the following key tasks:

1. **Tensor Operations:**
   a. Created a random tensor (**a**) and explored basic tensor operations, such as shape printing and statistical analysis.
2. **Tensor Conversion:**
   a. Converted the tensor to a NumPy array and back, highlighting interoperability between PyTorch and NumPy.
3. **Statistical Analysis:**
   a. Calculated and printed mean, standard deviation, and median values of the tensor.
4. **Mean Computation:**
   a. Computed mean values along specified axes, demonstrating PyTorch's axis-wise operations.
5. **Neural Network Construction:**
   a. Defined a simple neural network (**model_tmp**) using PyTorch's neural network module (**nn.Module**).
6. **Tensor Flattening and Network Application:**
   a. Flattened the tensor and applied the neural network, showcasing tensor manipulation and network application.
7. **Loss Calculation:**
   a. Calculated the Mean Squared Error (MSE) loss between the network output and a target tensor.
8. **Backpropagation:**
   a. Executed backpropagation to compute gradients, emphasizing PyTorch's automatic differentiation.
9. **Optimization:**
   a. Utilized Stochastic Gradient Descent (SGD) optimization to update model parameters based on gradients, confirming the expected parameter variations.

This summary encapsulates your exploration of PyTorch's foundational functionalities, spanning tensor operations, neural network construction, loss computation, and optimization processes.

2. Simple fully-connected NN (MLP)

Model Definition:

- Created a simple fully-connected neural network (MLP) to classify flattened MNIST images. The MLP consists of two fully-connected layers with ReLU activation.

Training Setup:

- Chose Adam optimizer and CrossEntropyLoss as the criterion.
- Trained the model for 10 epochs on the MNIST dataset, keeping track of the best model based on validation loss.

Training Loop:

- Iterated through epochs, performing training and validation steps.
- Recorded and printed average training and validation losses at the end of each epoch.

Plotting Losses:

- Plotted the training and validation losses over epochs on the same graph.

Testing:

- Tested the best model on the test set and printed the test loss and accuracy.

Saving the Best Model:

- Saved the best model's state dictionary to a file named 'best_model.pth'.

This section involved the creation, training, evaluation, and saving of a simple MLP for MNIST digit classification using PyTorch.

3. Bonus - Weight game

3-1. Display Average Images:
Task: Display the average image of each label (digit 0 to 9) over the validation set.

- **Overview:**
    - For each digit label, extract images from the validation set with that label.
    - Calculate the average image for each digit.
    - Display the average images.

3-2. Build a 1-layer NN:
Task: Build a 1-layer neural network with Binary Cross Entropy Loss. Display weights with respect to labels before and after training.

- **Overview:**
- Create a simple 1-layer neural network with 28x28 input size and 10 output neurons.

    Before Training:
    - Display randomly initialized weights.
    - Print the validation loss at this moment.
- Train the model until the validation loss is less than 0.1.

    After Training:
    - Display the weights after training.
    - Print the validation loss after training.

3-3. Mechanism Summary:
Task: Summarize the mechanism with one word.

- **Overview:**
- **Linear** describe the underlying mechanism of the 1-layer neural network in solving the problem.

The display involves visualizing average images and weights. Training involves a simple 1-layer neural network with Binary Cross Entropy Loss. The mechanism summary is a conceptual overview of how the neural network is functioning for this specific problem.

4. CNN

Building a Convolutional Neural Network (CNN) for MNIST Classification:

1. **Model Architecture:**
   - A simple CNN called **SimpleCNN** is defined.
   - It consists of two convolutional layers with ReLU activation and max-pooling, followed by two fully connected layers.

2. **Data Loading and Preprocessing:**
   - MNIST dataset is loaded using PyTorch's **datasets** and **DataLoader**.
   - Transformations include conversion to tensor and normalization.

3. **Model Training:**
   - The CNN is trained for several epochs using a training loop.
   - Cross-Entropy Loss is used as the criterion for optimization.
   - Adam optimizer is employed for updating model parameters.
   - Training and validation losses are recorded at each epoch.

4. **Monitoring and Saving Best Model:**
   - The model with the lowest validation loss is saved.
   - Training and validation losses are plotted to visualize the training process.

5. **Model Evaluation on Test Set:**
   - The saved best model is loaded.
   - The test set is used to evaluate the model's performance.
   - Test loss, accuracy, and a confusion matrix are calculated and printed.

6. **Visualization of Results:**
   - Confusion matrix is displayed as a heatmap.
   - Training and validation losses are plotted over epochs.
   - The best model is saved for future use.

7. **Computational Considerations:**
   - The number of parameters in the model is printed.
   - GPU acceleration (CUDA) is utilized if available.

8. **Saving the Best Model:**
   - Saved the best model's state dictionary to a file named 'best_model.pth'.

Summary:

- The code demonstrates the entire pipeline of building, training, saving, and evaluating a CNN for the MNIST digit classification task. Emphasis is placed on achieving a good test accuracy with a relatively simple CNN architecture.