Building and Training a Simple Neural Network in PyTorch

Your task is to implement a simple neural network using PyTorch. You will initialize all weights and biases randomly, train the network using gradient descent, and visualize the network's predictions before and after training.

Step 1: Implement the Initial Neural Network with Random Weights and Biases

- Implement a neural network class similar to the one demonstrated in PyTorch.
- Initialize all the weights (w00, w01, w02, w11, w12, w13, w20) and biases (b00, b01, b02) with random values using the torch.rand() function.
- Use **Sigmoid** activation functions in the hidden layers and **Tanh** for the output layer.
- Use the following architecture:
  - Three input-to-hidden layers
  - One output layer
- **Deliverable**: A Python script that outputs the network's predictions before training. Plot the results using seaborn.

Step 2: Build the Trainable Neural Network

- Modify the network so that the weights and biases are trainable (requires_grad=True).
- Use the **Stochastic Gradient Descent (SGD)** optimizer and **Mean Squared Error (MSE)** loss function.
- **Deliverable**: A Python script that shows both the initial predictions before training and the updated predictions after training.

Step 3: Implement the Training Loop

- Implement a training loop that runs for 100 epochs.
- For each epoch, update the trainable parameters using the optimizer and calculate the loss using the true and predicted outputs.
- Print the total loss after each epoch.
- **Deliverable**: A script that prints the loss after each epoch to track progress.

Step 4: Visualization

- After training, plot the final predictions using seaborn.
- Compare the network's predictions before and after training using line plots.
- **Deliverable**: Two plots—one showing the network's output before training, and another showing the output after training.