Exam - I

Part A

Neural Network Architectures

1. Multi-Layer Perceptron (MLP): A basic form of neural network with multiple layers of neurons, each layer fully connected to the next one.
2. Convolutional Neural Network (CNN): Designed for processing data with a grid-like topology (e.g., images), using convolutional layers.
3. Deep Neural Network (DNN): A neural network with a significant number of layers, enabling complex representations of data.
4. Graph Neural Network (GNN): Specialized in processing data represented as graphs, capturing relationships and structures within the data.
5. Deep Q-Networks (DQNs): Combines Q-learning with deep neural networks, mainly used in reinforcement learning applications.
6. Transformers: An architecture primarily used in natural language processing, relying on self-attention mechanisms.
7. Recurrent Neural Network (RNN): Suitable for sequential data, with neurons that have loops to retain information over time.
8. Long Short Term Memory Network (LSTM): A type of RNN capable of learning long-term dependencies, often used in time-series analysis.
9. Recurrent Convolutional Network (RCN): Combines the spatial hierarchy of CNNs with the temporal processing capabilities of RNNs.
10. Hopfield Network: A recurrent neural network with binary threshold nodes, used in associative memory applications.
11. Autoencoder: Used for unsupervised learning of efficient codings, typically for dimensionality reduction.
12. Variational Autoencoder: An extension of autoencoders, mainly used for generative tasks.
13. Generative Adversarial Network (GAN): Consists of two networks, a generator and a discriminator, competing against each other.

14. Echo State Networks (Reservoir Computing Network): A recurrent network with a fixed, randomly connected hidden layer.
15. Graph Neural Network: Processes data structured as graphs, capturing their relational nature.

Dynamic Neural Networks

16. Adaptive Filters: Feed-forward dynamic networks that adapt their response based on input history.
17. Feedback (Recurrent) Networks: Networks like the Hopfield network, where the output depends on both current and previous inputs.

Learning Paradigms

18. Supervised Learning: The model learns from labeled data, mapping inputs to outputs.
19. Unsupervised Learning: Learning from unlabeled data, often discovering hidden patterns or structures.
20. Reinforcement Learning: Learning to make decisions by receiving rewards or penalties as feedback.

Part B

1.

The code can be found in the following link:

digit_recognition_custom_optim.ipynb

The results are shown below:

Epoch 1 - Training Loss: 0.4251, Validation Accuracy: 0.9374

Epoch 2 - Training Loss: 0.1819, Validation Accuracy: 0.9559

Epoch 3 - Training Loss: 0.1320, Validation Accuracy: 0.9609

Epoch 4 - Training Loss: 0.1050, Validation Accuracy: 0.9608

Epoch 5 - Training Loss: 0.0896, Validation Accuracy: 0.9604

Epoch 6 - Training Loss: 0.0783, Validation Accuracy: 0.9598

Epoch 7 - Training Loss: 0.0682, Validation Accuracy: 0.9722

Epoch 8 - Training Loss: 0.0627, Validation Accuracy: 0.9748

Epoch 9 - Training Loss: 0.0559, Validation Accuracy: 0.9691

Epoch 10 - Training Loss: 0.0501, Validation Accuracy: 0.9771

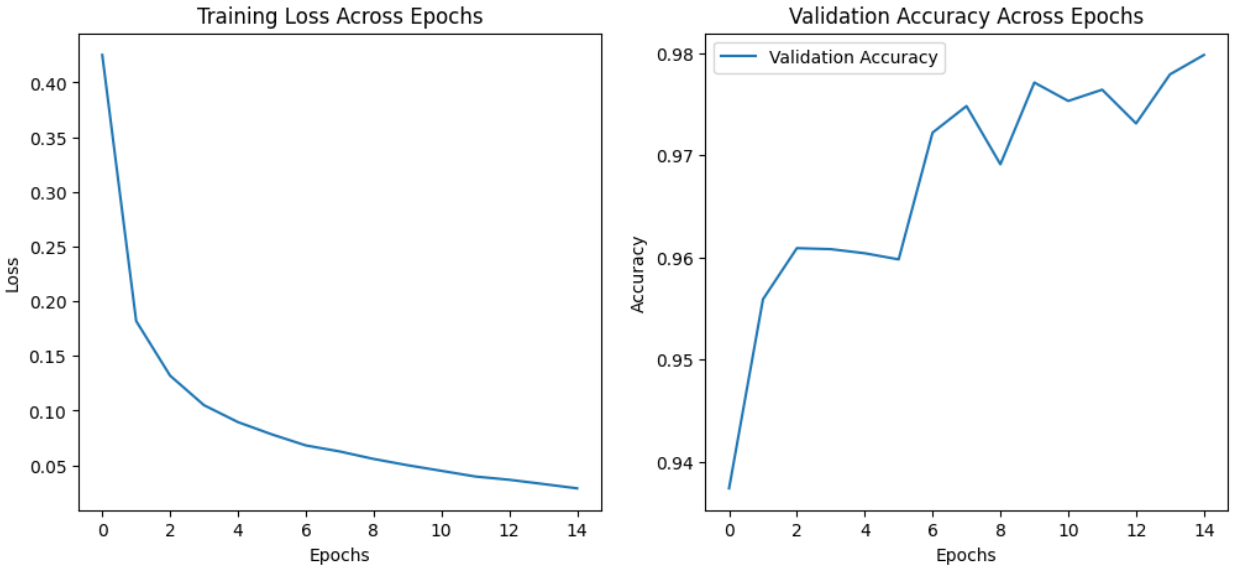
Epoch 11 - Training Loss: 0.0450, Validation Accuracy: 0.9753

Epoch 12 - Training Loss: 0.0398, Validation Accuracy: 0.9764

Epoch 13 - Training Loss: 0.0368, Validation Accuracy: 0.9731

Epoch 14 - Training Loss: 0.0330, Validation Accuracy: 0.9779

Epoch 15 - Training Loss: 0.0290, Validation Accuracy: 0.9798



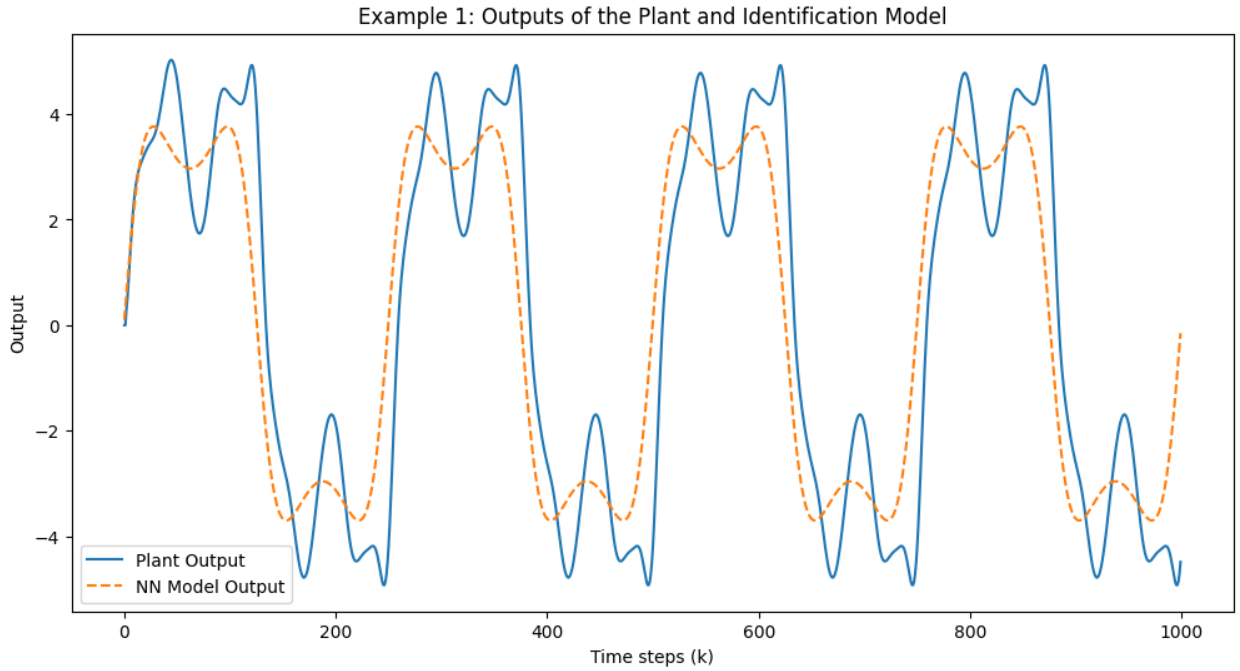
I believe the previous code adequately addresses the question as it doesn't rely on any optimization packages and solely utilizes the model parameters we've defined in PyTorch. However, if the intention behind the question was to create a comprehensive implementation entirely from scratch, you can find such code here.

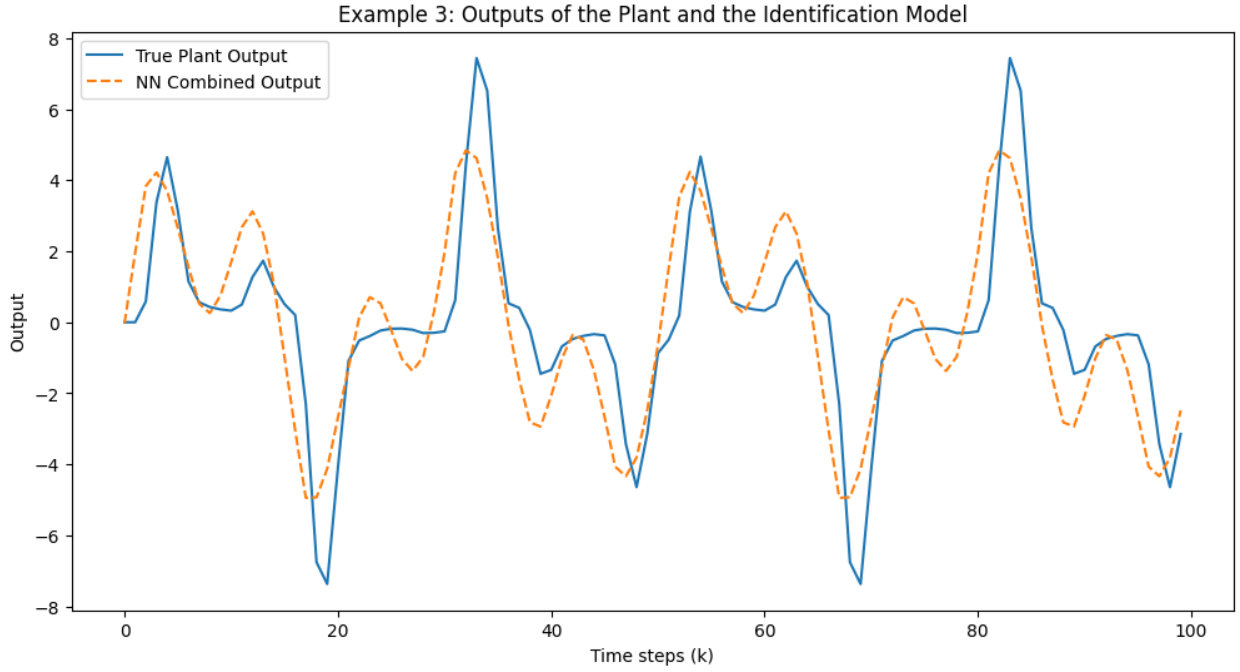
2.

The code can be found in the following link:

nn_dynamic_systems.ipynb

The results:





Example 1 Summary:

Problem: The goal is to identify a plant governed by a specific difference equation using a neural network.

Plant Formulation: The plant is defined by the equation

$$y_p(k+1) = 0.3y_p(k) + 0.6y_p(k-1) + f(u(k))$$

Where

$$f(u) = 0.6\sin(\pi u) + 0.3\sin(3\pi u) + 0.1\sin(5\pi u)$$

Neural Network Architecture: A single neural network is used for the identification model. This network approximates the unknown function f.

Training Process: The neural network is trained using backpropagation with a step size of 0.25. The input to both the plant and the model is a sinusoid

$$u(k) = \sin(2\pi k/250)$$

Training is conducted over a period with both regular and random inputs to ensure the model learns the plant's behavior accurately.

Example 3 Summary:

Problem: This example focuses on a more complex plant identification, where the input appears nonlinearly in the plant's difference equation.

Plant Formulation: The plant is described by a nonlinear model with the form

$$y_p(k+1) = f(y_p(k)) + g(u(k-1)),$$

where $f(y_p(k)) = \frac{y_p(k)}{1+y_p(k)^2}$ and $g(u(k)) = u(k)^3$.

Neural Network Architecture: Two separate neural networks, N_f, N_g, are used to approximate the functions f and f respectively. Each network consists of a layer with 20 neurons.

Training Process: The networks are trained using a step size of 0.1 over 100,000 time steps. The input u(k) for training is a random signal within the interval [-2, 2], ensuring

that g approximates the function over this range. The output y_p is then used to train N_f, ensuring it approximates f over the required range. For validation, a sinusoidal input is used to demonstrate the model's effectiveness.