MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

II Semester M. Tech Computer Science & Engineering

CSE 5245 Deep Learning & Applications Lab [0 0 3 1]

LABORATORY MANUAL

Department of Computer Science & Engineering
Manipal Institute of Technology, Manipal, India
January 2024

COURSE OUTCOMES (COs)

At the end of this course, the student should be able to:		No. of Contact Hours	Marks
CO1	Use the tools involved in building a deep neural network model.	9	30
CO2	Train and test deep learning models.	15	30
CO3	Apply deep learning models for solving real-life problems.	12	40
	Total	36	100

ASSESSMENT PLAN

Components	Continuous Evaluation	End semester Examination
Duration	3 Hours per week	150 Minutes
Weightage	60%	40%
Pattern	• 2 evaluations of 20 marks each: 2 * 20 = 40 marks, out of which: ▪ Record : 6 marks, ▪ Program execution : 7 marks, ▪ Quiz : 7 marks • 1 Mid-Sem Examination = 20 marks	Model Performance Analysis : 15 marks, Program execution : 25 marks.

Instructions to students:

1. Students must document their code, explaining the purpose of each component and function.
2. Provide comments within the code to enhance readability.
3. Write a report summarizing the training process, challenges faced, and results obtained.

Submission:

1. Students are required to submit their code in GitHub.
2. Optionally, present their findings in a short presentation, discussing insights and potential improvements, whenever asked.

LESSON PLAN

Week No	TOPICS	Course Outcome Addressed
Week 1	Getting Started with IDE, NumPy, Pandas, SciKit Learn, Matplotlib, Plotting, tensors and setting up working environment	CO1
Week 2	Implementation of Neural Networks	CO1
Week 3	Introduction to PyTorch and Tensors	CO1

Week 4	Implementation of Convolutional Neural Networks (CNNs)	CO2
Week 5	Comparing Convolutional Neural Networks (CNN) Vs Fully Connected Neural Networks for Image Classification	CO2
Week 6	Mid-Semester Examination	
Week 7	Implementation of Recurrent Neural Networks (RNNs)	CO2
Week 8	Implementation of LSTM	CO2
Week 9	Implementation of Transfer Learning	CO3
Week 10	Implementation of Image Reconstruction and Image Denoising Using Autoencoders	CO3
Week 11	Generative Adversarial Networks (GANs)	CO3
Week 12	End-term lab examination	

REFERENCES

1 Aurelien Geron, "Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow, OReilly Publications

2 François Chollet, "Deep learning with Python", Manning press 2021

3 Ian Goodfellow, Yoshua Bengio and Aaron Courville, " Deep Learning", MIT Press, 2017.

4 Josh Patterson, Adam Gibson "Deep Learning: A Practitioner's Approach", O'Reilly Media, 2017

5 www.machinelearningmastery.com

6 www.pyimagesearch.com

WEEK-1: Getting Started

Objective:
The primary goal of this lab is to familiarize students with the essential programming tools for deep learning, emphasizing Python programming and the NumPy library. By the end of this lab, students should have a solid foundation in using Python for numerical computations, which is crucial for implementing and understanding deep learning algorithms.

1. Installation of IDE and setting up working environment.
2. Brief overview of Python programming language.
3. Introduction to the NumPy library for numerical operations in Python.
4. Creating NumPy arrays and basic array operations.
5. Indexing and slicing arrays.
6. Simple numerical exercises to reinforce Python and NumPy concepts.
7. Implementing basic mathematical operations on arrays.

WEEK-2: Implementation of Neural Networks

Objective:
The primary goal of this lab is to provide students with a hands-on introduction to artificial neural networks (ANNs), the fundamental building blocks of deep learning. By the end of this lab, students should be familiar with basic neural network architectures, understand the components of a neural network, and be able to implement a simple neural network from scratch.

Tasks:

1. Introduction to Neural Networks:
   - Brief theoretical overview of artificial neural networks.
   - Explanation of key terms: neurons, layers, weights, biases, activation functions.
2. Building a Simple Neural Network from Scratch:
   - Implement a basic neural network with a single hidden layer using NumPy.
   - Define the input layer, hidden layer, and output layer.
   - Initialize random weights and biases.
   - Implement the forward pass using a sigmoid activation function.
   - Calculate the loss using a simple mean squared error.
3. Training the Neural Network:
   - Generate a synthetic dataset for a binary classification problem.
   - Split the dataset into training and testing sets.
   - Implement the backpropagation algorithm to update weights and biases during training.
   - Iterate through epochs and observe how the loss decreases.
4. Evaluation:
   - Use the trained neural network to make predictions on the test dataset.
   - Evaluate the performance using metrics such as accuracy.
   - Visualize the decision boundary learned by the neural network.

5. Experimentation:
    - Encourage students to experiment with different architectures (number of hidden layers, neurons per layer).
    - Explore the impact of changing hyperparameters (learning rate, activation functions).

WEEK-3: Introduction to PyTorch and Tensors

Objective:
The primary objective of this lab is to introduce students to PyTorch, a popular deep learning framework, and familiarize them with tensors, the fundamental data structure in PyTorch. By the end of this lab, students should be comfortable using PyTorch for basic operations and understand the role of tensors in deep learning.
Introduction to PyTorch:

1. Brief overview of PyTorch and its significance in the deep learning ecosystem.
2. Comparison with other deep learning frameworks.
3. Installation and setup of PyTorch.

Understanding Tensors:

1. Explanation of tensors as the core data structure in PyTorch.
2. Differentiating between scalars, vectors, matrices, and higher-dimensional tensors.
3. Creation of tensors using PyTorch.

Basic Tensor Operations:

1. Performing basic operations on tensors (addition, subtraction, multiplication).
2. Utilizing built-in functions for common mathematical operations.
3. Discussing the concept of in-place operations and their implications.

Tensor Indexing and Slicing:

1. Understanding tensor indexing and slicing.
2. Extracting sub-tensors from a larger tensor.
3. Demonstrating the similarities with NumPy array operations.

Gradient Computation with Autograd:

1. Introduction to automatic differentiation in PyTorch using the Autograd module.
2. Enabling gradient computation for tensors.
3. Calculating gradients for simple functions.

WEEK-4: Implementation of Convolutional Neural Networks (CNNs)

Objective:
The primary goal of this lab is to provide students with hands-on experience in understanding, implementing, and training Convolutional Neural Networks (CNNs). By the end of this lab, students should be proficient in the basics of CNN architecture, feature extraction, and its application to image classification tasks.

Consider the Fashion MNIST dataset [Fashion MNIST dataset, an alternative to MNIST] and do the following:

Understanding the Dataset and Pre-processing: Implement the following:

1. Compute and display the number of classes.
2. Compute and display the dimensions of each image.
3. Display one image from each class.
4. Perform normalization.

Performing experiments on Fully Connected Neural Networks (FCNN):

1. Design a Fully CNN which is most suitable for the given dataset: Experimentally choose the best network
2. Train and test the network (choose the best epoch size so that there is no overfitting). Plot the performance curves.

WEEK-5: Comparing Convolutional Neural Networks (CNN) Vs Fully Connected Neural Networks for Image Classification

Objective:
The primary goal of this lab is to provide students with hands-on experience in understanding, implementing, and training Convolutional Neural Networks (CNNs). By the end of this lab, students should be proficient in the basics of CNN architecture, feature extraction, and compare performance of FCNN and CNN for image classification tasks.

Performing experiments on a Convolutional Neural Networks (CNNs):

1. Design CNN-1 which contains:
   a. One Convolution layer which uses 32 kernels each of size 5x5, stride = 1 and, padding =0.
   b. One Pooling layer which uses MAXPOOLING with stride =2.
   c. One hidden layer having number of neurons = 100
2. Design CNN-2 which contains:
   a. Two back-to-back Convolution layers which uses 32 kernels each of size 3x3, stride = 1, and padding =0.
   b. One Pooling layer which uses MAXPOOLING with stride =2.
   c. One hidden layer having number of neurons = 100

   Note: use ReLU activation function after each convolution layer.
3. Train and test the networks (choose the best epoch size so that there is no overfitting).
4. Plot the performance curves for CNN-1 and CNN-2.
5. Compare the performances of CNN-1 and CNN-2.

Compare the performances of FCNN and CNN.
Compare the number of parameters in the FCNN and the CNN.
Discuss the computational efficiency of both networks. Which one took longer to train and why?

WEEK-6: Midterm test (20 marks)

Dataset and task will be provided to the students. They must code, analyse and present the results.

WEEK-7: Implementation of Recurrent Neural Networks (RNNs)

Objective:
The primary goal of this lab is to provide students with hands-on experience in understanding, implementing, and training Recurrent Neural Networks (RNNs). By the end of this lab, students should be proficient in the basics of RNN architecture, sequential data processing, and its application to tasks involving sequences.

Building a Basic RNN:
1. Implementing a simple RNN using PyTorch or TensorFlow.
2. Understanding the architecture of RNN cells and their recurrent nature.
3. Training the RNN on a simple sequential dataset (e.g., time-series data).

Applying RNNs to Time-Series Prediction:
1. Exploring the application of RNNs in time-series prediction tasks.
2. Implementing an RNN for predicting future values in a time-series dataset.
3. Discussing challenges and strategies for improving predictive performance.

Ref: https://machinelearningmastery.com/lstm-for-time-series-prediction-in-pytorch/

WEEK-8: LSTM

Objective:
The primary goal of this lab is to provide students with hands-on experience in understanding, implementing, and training LSTM. By the end of this lab, students should be proficient in the basics of LSTM its application to tasks involving sequences.

1. Build LSTMs for Human Activity Recognition Time Series Classification
(Ref: https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recognition-time-series-classification/)

2. Work on Text Generation With LSTM Recurrent Neural Networks

WEEK-9: Implementation of Transfer Learning

Objective:
The primary goal of this lab is to provide students with hands-on experience in understanding, implementing, and applying transfer learning techniques in deep learning. By the end of this lab, students should be proficient in utilizing pre-trained models and fine-tuning them for specific tasks.

Fine-Tuning a Pre-trained Model:
1. Introduction to fine-tuning: adapting a pre-trained model to a specific problem.
2. Implementing fine-tuning using PyTorch.

3. Discussing considerations for selecting and modifying layers during fine-tuning.

Transfer Learning for Image Classification:

1. Applying transfer learning to an image classification task.
2. Utilizing a pre-trained model (e.g., ResNet, VGG, or Inception) from a deep learning framework's model zoo.
3. Fine-tuning the model on a new dataset and evaluating its performance.

Transfer Learning for Object Detection:

1. Extending transfer learning to object detection tasks.
2. Implementing transfer learning with a pre-trained model for object detection (e.g., using Faster R-CNN or YOLO).

WEEK-10: Implementation of Image Denoising Using Autoencoders

Objective:
The primary goal of this lab is to provide students with a hands-on understanding of autoencoders, a type of neural network used for unsupervised learning and dimensionality reduction. Students will explore different variants of autoencoders and their applications.

Vanilla Autoencoder Implementation:

1. Implementation of a basic vanilla autoencoder using PyTorch or TensorFlow.
2. Training the autoencoder on a simple dataset (e.g., MNIST digits) for reconstruction.
3. Visualization and comparison of the original and reconstructed images.

Denoising Autoencoder Implementation:

1. Understanding the concept of denoising autoencoders.
2. Implementation of a denoising autoencoder using PyTorch.
3. Training the autoencoder to reconstruct clean images from noisy inputs.
4. Evaluation of the model's performance in denoising.

WEEK-11: Generative Adversarial Networks (GANs)

Objective:
The primary objective of this lab is to provide students with hands-on experience in understanding, implementing, and training Generative Adversarial Networks (GANs). By the end of this lab, students should be proficient in building GANs for generating synthetic data.

Building a Basic GAN:

1. Implementation of a basic GAN using PyTorch or TensorFlow.
2. Training the GAN on a simple dataset (e.g., MNIST digits) for generating realistic images.
3. Visualizing and evaluating the generated samples.

(Ref: https://machinelearningmastery.com/cyclegan-tutorial-with-keras/)