Department of Computer Science

Lab Manual

of

DEEP LEARNING

MAI417-3

Class Name: III MScAIML

Master of Science Artifitial Intelligence and Machine Learning

2025-26

Prepared by: Huda Maniyar and Ms Shivangi

Verified by: Ms Helen

Department Overview

Department of Computer Science of CHRIST (Deemed to be University) strives to shape outstanding computer professionals with ethical and human values to reshape nation's destiny. The training imparted aims to prepare young minds for the challenging opportunities in the IT industry with a global awareness rooted in the Indian soil, nourished and supported by experts in the field.

Vision

The Department of Computer Science endeavours to imbibe the vision of the University "**Excellence and Service"**. The department is committed to this philosophy which pervades every aspect and functioning of the department.

Mission

"To develop IT professionals with ethical and human values". To accomplish our mission, the department encourages students to apply their acquired knowledge and skills towards professional achievements in their career. The department also moulds the students to be socially responsible and ethically sound.

Introduction to the Programme

Machines are gaining more intelligence to perform human like tasks. Artificial Intelligence has spanned across the world irrespective of domains. MSc (Artificial Intelligence and Machine Learning) will enable to capitalize this wide spectrum of opportunities to the candidates who aspire to master the skill sets with a research bent. The curriculum supports the students to obtain adequate knowledge in the theory of artificial intelligence with hands-on experience in relevant domains with tools and techniques to address the latest demands from the industry. Also, c andidates gain exposure to research models and industry standard application development in specialized domains through guest lectures, seminars, industry offered electives, projects, internships, etc.

Programme Objective

- To acquire in-depth understanding of the theoretical concepts in Artificial Intelligence and Machine Learning
- To gain practical experience in programming tools for Data Engineering, Knowledge Representation, Artificial intelligence, Machine learning, Natural Language Processing and Computer Vision.
- To strengthen the research and development of intelligent applications skills through specialization based real time projects.
- To imbibe quality research and develop solutions to the social issues.

Programme Outcomes:

PO1 : Conduct investigation and develop innovative solutions for real world problems in industry and research establishments related to Artificial Intelligence and Machine Learning

PO2 : Apply programming principles and practices for developing automation solutions to meet future business and society needs.

PO3 : Ability to use or develop the right tools to develop high end intelligent systems
PO4 : Adopt professional and ethical practices in Artificial Intelligence application development
PO5 : Understand the importance and the judicious use of technology for the sustainability of the environment.

MAI417-3– Deep Learning

Total Teaching Hours for Trimester: 75
Max Marks: 100
Credits: 4

Course Description

The course introduces deep learning techniques, focusing on feedforward networks, CNNs, RNNs, and autoencoders. Students will learn regularization methods, optimization for long-term dependencies, and advanced model implementations. Lab exercises provide hands-on experience with Keras and TensorFlow. By the end, students will apply these techniques to real-world problems in computer vision and sequence processing.

Course Objectives

The main objective of this course is to make students comfortable with the tools and techniques required to handle large datasets. Several libraries and datasets publicly available will be used to illustrate the application of these algorithms. This will help students develop the skills required to gain experience in doing independent research and study.

Course Outcomes

Upon successful completion of the course, the student will be able to

CO1: Implement deep feedforward and backpropagation networks for classification.
CO2: Analyze regularization techniques for optimizing deep learning models.
CO3: Apply convolutional and recurrent neural networks for vision and sequence processing tasks.
CO4: Evaluate autoencoders for feature extraction and data compression.
CO5: Compare different deep learning architectures for performance improvement.

Unit-1 **Teaching Hours: 15**

DEEP FEEDFORWARD NETWORKS

An overview of ANN, Back Propagation Neural Networks, Deep Feedforward Networks: Deep network for Universal Boolean function representation, Classification and Approximation, Perceptron Learning.

Lab Exercises:

1. Demonstrate MLP in Keras/TensorFlow
2. Demonstrate Deep Feedforward Network

Unit-2 **Teaching Hours: 15**

REGULARIZATION FOR DEEP MODELS

Regularization for Deep models: L2 and L1 Regularization, Constrained Optimization and Under- Constrained Early Stopping, Parameter Tying and Parameter Sharing, Sparse representations, Dropout

Lab Exercises:

3. Demonstrate Regularization L1 and L2 for Deep learning model

Unit-3 **Teaching Hours: 15**

CONVOLUTIONAL NEURAL NETWORK

Introduction to convolution neural networks: stacking, striding and pooling, Applications in Computer Vision

Lab Exercises:

4. Demonstrate Convolution Neural Network

Unit-4 **Teaching Hours: 15**

RECURRENT NEURAL NETWORKS

Sequence Processing, Unfolding Computational Graphs, Training recurrent networks

The Long Short-Term Memory (LSTM), Optimization for Long- Term Dependencies, Encoder-Decoder Sequence-to-Sequence processing

Lab Exercises:

5. Demonstrate Short-Term Long Memory (LSTM)

Unit-5 **Teaching Hours: 15**

AUTOENCODERS

The architecture of autoencoders - the relationship between the Encoder, Bottleneck, and Decoder, how to train autoencoders? Types of autoencoders: Undercomplete autoencoders, Sparse autoencoders, Contractive autoencoders, Denoising autoencoders, Variational Autoencoders

Lab Exercises:

6. Demonstrate Sparse Autoencoders
7. Demonstrate Contractive Autoencoders

Text Books and Reference Books

[1] Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville. MIT Press 2016.

[2] Simon J.D. Prince, Understanding Deep Learning: The Simple Math of Deep Learning, MIT Press 2024.

Essential Reading / Recommended Reading

[1] Deep Learning with Python by Francois Chollet. 2nd Edition, Manning Publications Co., 2020

[2] Introduction to Deep Learning by Eugene Charniak. The MIT Press 2019

[3] Dive into Deep Learning by Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola. 2019

[4] AurélienGéron, “Hands-On Machine Learning with Scikit- Learn and TensorFlow”, O’Reilly, 2022.

Web Resources:

[1] https://www.deeplearningbook.org/

[2] https://archive.ics.uci.edu/ml/datasets.php

CO – PO Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7
CO1	3	3	2	2	2	1	2
CO2	2	2	1	1	1	1	2
CO3	3	3	2	1	2	1	2
CO4	2	2	1	1	1	1	2
CO5	3	3	2	2	2	2	3

LIST OF PROGRAMS

MSCAIML

Sl. no	Title of lab Experiment	Page number	RBT	CO
1	Learning the XOR Boolean Function Using an MLP		L3	CO1, 3
2	Deep Feedforward Neural Network for Fashion MNIST Classification		L3	CO2, 3
3	Effect of L1, L2, and Elastic Net Regularization on Model Training		L3	CO3
4			L3	CO3
5			L3	CO3
6			L3	CO3, 4
7			L4	CO3
8			L3	CO3
9			L3	CO4
10			L5	CO4

Evaluation Rubrics:

Evaluation Rubrics for each program would be

Correctness and Demonstration and timely submission (5 =3+2marks)
Concept Clarity and modeling (3 marks)
Initiative & Effort ((2 marks)

Submission Guidelines:

- Make a copy of the lab manual template with your <name_reg:no_subject name >,
- Copy the given question and the answer (lab code) with results, followed by the conclusion of that lab. Title the lab as lab number.
- Keep updating your lab manual and show the lab manual of that particular lab for evaluation.

- Create a Git Repository in your profile <SPR lab-reg no> . Follow a different branch for each lab <Lab 1, Lab 2…>, and push the code to Git. The link should be provided in Google Classroom along with the PDF of the lab manual.
- Upload the PDF to Google Classroom before the deadline.

Lab 1

Lab Exercise I: Learning the XOR Boolean Function Using an MLP

Aim

1. To understand how to implement neural networks using different deep learning libraries (**Keras, PyTorch, and TensorFlow**).
2. To solve the non-linear XOR problem using an MLP and study the effect of hyperparameters such as learning rate, activation functions, number of neurons, and epochs on model performance.

Question

Implement an MLP to learn the XOR Boolean function

The XOR function takes two binary inputs (0 or 1) and produces a binary output (0 or 1) based on the following rule:

Input 1	Input 2	XOR Output
0	0	0
0	1	1
1	0	1
1	1	0

Steps:

1. **Create the Dataset**
    - Define input (X) and output (y) arrays for all 4 XOR combinations.
2. **Build an MLP**
    - Input layer: size 2 (for the two inputs).
    - Hidden layer: at least 2 neurons with **ReLU** or **Tanh** activation.
    - Output layer: 1 neuron with **sigmoid** activation for binary classification.
3. **Compile the Model / Define Loss and Optimizer**
    - Use **Binary Cross-Entropy** loss.
    - Use an optimizer of your choice (e.g., **Adam** or **SGD**).
4. **Train the Model**

   - Train the model on the XOR dataset.
   - Experiment with **epochs, learning rate, and number of neurons** to improve performance.
5. **Evaluate the Model**
   - Predict outputs for all 4 input combinations.
   - Check if the network correctly learns the XOR function.
6. **Implement Using Three Libraries**
   - Repeat the above steps using:
     1. **Keras (TensorFlow high-level API)**
     2. **PyTorch**
     3. **TensorFlow low-level API**

Additional Exercises (Optional):

- Plot the decision boundary for each implementation.
- Compare training curves and final accuracy between libraries.
- Discuss how changes in **learning rate, activation function, hidden layers, or epochs** affect learning.

Evaluation Rubrics

1. **Implementation** – 5 marks
2. **Complexity and Validation** – 3 marks
3. **Documentation & Writing the inference** – 2 marks

Submission Guidelines

1. Make a copy of the lab manual template with your <name_reg:no_subject name>
2. Copy the given question and the answer (lab code) with results, followed by the conclusion of that lab. Title the lab as Lab 1.
3. Keep updating your lab manual and show the lab manual of that particular lab for evaluation.
4. Create a **Git repository** in your profile <SPR lab-reg no>. Follow a different branch for each lab <Lab 1, Lab 2 …> and push the code to Git.
5. Provide the Git link in **Google Classroom** along with the PDF of the lab manual.
6. Upload the PDF to Google Classroom before the deadline.

Code with Results

Libraries Used

- Keras (TensorFlow high-level API)
- PyTorch
- TensorFlow (low-level API)

Dataset

The XOR dataset consists of four input combinations. These values were stored in arrays `X` and `y` and used for training in all three implementations.

```
# Dataset
X = np.array([[0, 0],[0, 1],[1, 0],[1, 1]], dtype=np.float32)
y = np.array([[0], [1], [1], [0]], dtype=np.float32)
```

Model Architecture (Common for all)

- Input layer: 2 neurons
- Hidden layer: 4 neurons
- Activation: ReLU (Keras) / Tanh (PyTorch & TensorFlow)
- Output layer: 1 neuron
- Activation: Sigmoid
- Loss Function: Binary Cross-Entropy
- Optimizer: Adam
- Epochs: 500

Results

1. Keras Implementation

```
predictions = model.predict(X)
print("Predicted Outputs:")
print(np.round(predictions))
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 113ms/step
Predicted Outputs:
[[0.]
 [1.]
 [1.]
 [0.]]
```

2. PyTorch

```
with torch.no_grad():
    predictions = model(X_torch)
    print(torch.round(predictions))
```

```
tensor([[0.],
        [1.],
        [1.],
        [0.]])
```

3. Tensorflow

```
print(tf.round(model(X)))
```

```
tf.Tensor(
[[0.]
 [1.]
 [1.]
 [0.]], shape=(4, 1), dtype=float32)
```

Decision Boundary

For all three implementations:

The decision boundary clearly separates:

- Class 0 → (0,0) and (1,1)
- Class 1 → (0,1) and (1,0)

This confirms that the MLP learned a non-linear decision surface, which is required for XOR.

Conclusion

In this experiment, a Multi-Layer Perceptron (MLP) was successfully implemented to learn the XOR Boolean function using three different deep learning libraries: Keras, PyTorch, and TensorFlow (low-level API).

The XOR problem is not linearly separable, which means it cannot be solved using a single-layer perceptron. By introducing a hidden layer with non-linear activation functions such as ReLU and Tanh, the network was able to model the complex relationship between inputs and outputs.

All three implementations achieved 100% accuracy on the XOR dataset, correctly predicting the output for all four input combinations.

Inference

- Need for Hidden Layers
  - The experiment proves that hidden layers are essential for solving non-linear problems like XOR. A single-layer model fails, while a multi-layer network succeeds.
- Effect of Activation Functions
  - Non-linear activations such as Tanh and ReLU enable the network to learn complex patterns. Without these, the model would behave like a linear classifier.
- Effect of Learning Rate and Epochs
  - Higher epochs improve convergence.
  - Very high learning rates may cause unstable training.
  - Adam optimizer provides fast and stable learning.
- Library Comparison
  - Keras: Easiest and fastest to implement.
  - PyTorch: Gives full control over training and debugging.

  - TensorFlow (low-level): Best for understanding internal working of neural networks.
- Overall Outcome
  - The experiment successfully demonstrates how neural networks overcome the limitations of linear models and highlights the importance of hidden layers in deep learning.

Lab 2

Lab Exercise: Deep Feedforward Neural Network for Fashion MNIST Classification

Aim

To design, implement and evaluate a deep feedforward neural network for classifying Fashion MNIST images using PyTorch, and to understand the role of depth, nonlinear activation, loss computation and gradient based learning in a supervised classification task.

Task

1. Load and preprocess Fashion MNIST data using PyTorch data loaders.
2. Construct a multilayer feedforward neural network with at least three hidden layers.
3. Use ReLU activation in hidden layers and linear outputs for classification.
4. Train the network using cross entropy loss and an optimizer such as Adam or SGD.
5. Monitor training loss and accuracy across epochs.
6. Evaluate the model on unseen test data and compute accuracy.
7. Explain the role of forward pass, backward pass and gradient updates.
8. Interpret the impact of hyperparameters such as learning rate, batch size and number of epochs.

Advanced Task 1: Experiment with Network Depth and Width(optional)

- Train models with different numbers of hidden layers (e.g., 1, 3, 5) and different neurons per layer.
- Compare training loss, test accuracy, and convergence speed.
- Discuss the trade-off between depth and overfitting.

Advanced Task 2: Activation Function Study

- Replace ReLU with Sigmoid, Tanh, and Leaky ReLU.
- Compare training time, convergence, and test accuracy.
- Explain why some activations perform better than others for deep networks.

Advanced Task 3: Visualization of Hidden Layers

- Visualize activations of hidden layers for a few sample images.
- Explain how features are transformed at each layer.

Evaluation Rubrics

1.

Rubrics	Marks
Correctness and Demonstration	5 marks
Concept Clarity (Viva)	3 marks
Initiative & Effort (self-learning)	2 marks

Submission Guidelines

1. Make a copy of the lab manual template with your <name_reg:no_subject name>
2. Copy the given question and the answer (lab code) with results, followed by the conclusion of that lab. Title the lab as Lab 1.
3. Keep updating your lab manual and show the lab manual of that particular lab for evaluation.
4. Create a **Git repository** in your profile <C lab-reg no>. Follow a different branch for each lab <Lab 1, Lab 2 …> and push the code to Git.
5. Provide the Git link in **Google Classroom** along with the PDF of the lab manual.
6. Upload the PDF to Google Classroom before the deadline.

STEPS

1. Implementation (Code)

Step 1: Import Libraries and Load Data We initialize PyTorch, define the transformation (converting images to tensors), and create DataLoaders for the FashionMNIST dataset.

Step 2: Define the Deep Feedforward Network We construct a Multi-Layer Perceptron (MLP) with 3 hidden layers (128, 64, 32 neurons) using ReLU activation.

Step 3: Training the Model The network is trained for 50 epochs using Backpropagation.

Step 4: Advanced Tasks (Depth and Activation Study) We compare different network depths (1, 3, 5 layers) and activation functions (ReLU, Sigmoid, Tanh, LeakyReLU).

2. Results and Visualizations

- **Final Training Accuracy:** ~95.76% (Epoch 50)
- **Test Accuracy:** 89.13%
- **Activation Study:**
  - **ReLU / LeakyReLU:** Fast convergence (~87% accuracy in 3 epochs).
  - **Sigmoid:** Slower initial learning due to vanishing gradients (~71% in Epoch 1).
  - **Tanh:** performed surprisingly well (~87%).

3. Inference

1. **Learning Progress:** The training loss decreased consistently from **594.11** to **104.41**, while accuracy rose from **77.66%** to **95.76%**. This confirms that the network successfully minimized the Cross-Entropy loss using the Adam optimizer and Backpropagation.
2. **Overfitting:** A gap was observed between Training Accuracy (95.7%) and Test Accuracy (89.1%). This indicates **overfitting**, where the model began memorizing specific training examples rather than generalizing perfectly to unseen data.
3. **Role of Activations:**
   - **ReLU** allows neurons to activate sparsely (outputting zero for negative inputs), which mitigates the vanishing gradient problem and accelerates convergence.
   - **Sigmoid** squashes outputs into a [0,1] range, causing gradients to become very small (vanish) during backpropagation, leading to slower learning.
4. **Feature Extraction:** The visualization of the hidden layer showed that some neurons respond strongly (high peaks) to specific image features, while others remain inactive (zero values due to ReLU), demonstrating the network's ability to isolate important visual patterns.

Lab 3

Lab Exercise: Effect of L1, L2, and Elastic Net Regularization on Model Training

Aim

To study the effect of L1, L2, and Elastic Net regularization on model training by analyzing loss, cost, accuracy, and changes in model weights.

Objectives

1. To train a baseline model without regularization and compute loss, cost, and accuracy.
2. To apply L1 regularization and analyze its impact on loss, cost, accuracy, and weight sparsity.
3. To apply L2 regularization and analyze its impact on loss, cost, accuracy, and weight magnitude.
4. To evaluate Elastic Net regularization and study its combined effect on sparsity and weight stability.
5. To compare how different regularization techniques influence overfitting and generalization using cost and weight analysis.

Tasks

1. Load a suitable publicly available dataset for classification.(for example, MNIST Handwritten Digits or Breast Cancer Wisconsin dataset).
2. Train a baseline model without regularization and record
   training and validation loss
   cost function value
   accuracy
   weight distribution
3. Train the model using L1 regularization and record
   training and validation loss
   cost function value including L1 penalty
   accuracy

number of weights driven to zero

4. Train the model using L2 regularization and record
   training and validation loss
   cost function value including L2 penalty
   accuracy
   change in weight magnitudes

5. Train the model using Elastic Net regularization and record
   training and validation loss
   total cost function value
   accuracy
   balance between sparsity and weight shrinkage

6. Compare all models based on
   loss versus cost behavior
   training and validation accuracy
   weight sparsity and magnitude

7. Interpret the results and justify the most suitable regularization technique for the given dataset.

Advanced Task (optional) – Hyperparameter Sensitivity

Investigate how varying the regularization strength (λ\lambdaλ) affects training loss, validation loss, weight magnitude, and sparsity for L1, L2, and Elastic Net. Plot the trends and analyze the trade-off between underfitting and overfitting.

1. Interpretation & Inference (For your Lab Manual)

This section analyzes the results obtained from the four different models (Baseline, L1, L2, Elastic Net).

A. Baseline Model (No Regularization)

- **Observation:** The baseline model achieved high accuracy on the training set. The weight distribution histogram shows a wide spread of values, with some weights having relatively large magnitudes (e.g., greater than ±0.5 or ±1.0).
- **Inference:** Without regularization, the model is free to assign large weights to specific features to minimize the training error. While this results in low training loss, large weights can indicate **overfitting**, where the model becomes overly sensitive to noise in the input data. The model is learning the training data "too well" but might struggle if the data distribution shifts slightly.

B. L1 Regularization (Lasso)

- **Observation:** The weight histogram for the L1 model is sharply peaked at zero. Many weights in the first layer have become exactly **0.0** (or extremely close to it).
- **Inference:** L1 Regularization adds a penalty proportional to the *absolute value* of the weights. This creates a "diamond-shaped" constraint that forces the optimization process to drive irrelevant feature weights to exactly zero.
- **Key Insight:** This demonstrates **Feature Selection**. The model effectively "turned off" the input features (columns in the breast cancer dataset) that were not helpful for classification, resulting in a **Sparse Model**. This makes the model more interpretable but can sometimes underfit if the regularization strength (λ) is too high.

C. L2 Regularization (Ridge)

- **Observation:** The weight histogram shows a "Bell Curve" (Gaussian) distribution centered around zero. Unlike L1, very few weights are exactly zero, but almost all weights are very small (e.g., between -0.1 and 0.1).
- **Inference:** L2 Regularization adds a penalty proportional to the *square* of the weights. This discourages any single weight from becoming too large but does not force them to zero.
- **Key Insight:** This creates **Weight Decay**. By keeping all weights small and diffuse, the model prevents any single neuron from dominating the decision. This reduces variance and makes the model robust against collinearity (correlated features), which is common in medical datasets.

D. Elastic Net Regularization

- **Observation:** The weights show characteristics of both techniques: many weights are pushed near zero (L2 effect), and some are exactly zero (L1 effect).
- **Inference:** Elastic Net combines the penalties. It enjoys the feature selection benefits of L1 while maintaining the stability of L2.
- **Key Insight:** This is often the most robust method for datasets like Breast Cancer Wisconsin, where groups of features might be correlated. Unlike L1 (which might pick just one feature from a correlated group and drop the rest), Elastic Net can keep the group together while still shrinking coefficients.

2. Conclusion

In this experiment, we successfully implemented and analyzed the effects of L1, L2, and Elastic Net regularization on a Deep Feedforward Neural Network using the Breast Cancer Wisconsin dataset.

1. **Sparsity vs. Shrinkage:** L1 regularization successfully produced a **sparse model** by driving weights to zero, effectively performing automatic feature selection. L2 regularization resulted in **weight shrinkage**, constraining the magnitude of weights to prevent overfitting without removing features.
2. **Performance:** While the Baseline model trained fastest, the regularized models (particularly L2 and Elastic Net) demonstrated more stable weight distributions. For this specific dataset, **L2 Regularization** provided the best balance between maintaining accuracy and preventing extreme weight values, as the dataset is small and sensitive to overfitting.
3. **Generalization:** The addition of penalty terms to the cost function increased the "Cost" value during training but ultimately led to a model that is less likely to memorize noise, ensuring better generalization to unseen test data.