

Vth Semester B. Tech Data Science & Engineering DSE 3141 Deep Learning Lab [0 0 3 1]

**LABORATORY MANUAL**

***Instructors*: Dr. Rohini Rao,**

**Dr. Abhilash K Pai, Dr. Gangothri,**

**Dr. Rajesh Mahadeva,**

**Dr. Namratha Marium Chacko**

**School of Computer Engineering, Manipal Institute of Technology, Manipal, India**

**JULY 2025**

**COURSE OUTCOMES (COS)**

|  |  |  |  |
| --- | --- | --- | --- |
| **At the end of this course, the student should be able to:** | | **No. of Contact Hours** | **Marks** |
| **CO1** | Apply the tools, on different dataset types, do performance evaluation methods, and fine-tuning strategies to build and optimize vanilla deep neural network models for performing classification and regression on structured data. | 7 | 15 |
| **CO2** | Design, develop, fine-tune, evaluate simple and advanced CNN models for Image classification. | 9 | 35 |
| **CO3** | Design, develop, fine-tune, evaluate simple and advanced RNN models for sequence modelling tasks like Time series prediction and NLP. | 12 | 35 |
| **CO4** | Design, develop, fine-tune, and evaluate Autoencoders and Generative models for representational learning. | 8 | 15 |
|  | **Total** | **36** | **100** |

**ASSESSMENT PLAN**

|  |  |  |
| --- | --- | --- |
| **Components** | **Continuous Evaluation** | **End semester Examination** |
| **Duration** | 2.5 Hours per week | 180 Minutes |
| **Weightage** | 60% | 40% |
| **Pattern** | * 1 evaluation of 20 marks:   1. Record : 6M,   2. Program execution : 7M,   3. Quiz : 7M * 1 Mid-Sem Examination: 20 marks * Mini Project : 20 marks   1. Phase1: Problem + Literature: 5M   2. Phase 2: End-to-End solution: 8M   3. Phase 3:Deployment & Demo: 7M | Model Performance Analysis: 15 marks,  Program execution : 25 marks. |

**LESSON PLAN**

|  |  |  |
| --- | --- | --- |
| **Week No** | **TOPICS** | **Course Outcome Addressed** |
| Week 1 | Tensorflow & Keras Tutotial,  Getting Started with Building Fully Connected Neural Networks In Keras | CO1 |
| Week 2 | Experimenting with Deep Neural Networks | CO1 |
| Week 3 | Convolutional Neural Networks (CNN) Vs Fully Connected Neural Networks for Image Classification | CO2 |
| Week 4 | Advanced CNN Architectures and Transfer Learning for Image Classification | CO2 |
| Week 5 | Recurrent Neural Networks for Time Series Forecasting | CO3 |
| Week 6 | Mid-Semester Examination, Mini Project | CO1, CO2, CO3 |
| Week 7 | LSTM and GRU for Sentiment Analysis | CO3 |
| Week 8 | Neural Machine Translation using Encoder-Decoder Architecture, Mini Project Phase 2 evaluation | CO3 |
| Week 9 | Image Reconstruction and Image Denoising Using Autoencoders, Mini Project Implementation | CO4 |
| Week 10 | Image Generation Using Generative Adversarial Networks, Mini Project Implementation | CO4 |
| Week 11 | Mini Project Final Evaluation | CO2, CO3, CO4 |
| Week 12 | End-term lab examination | CO2, CO3, CO4 |

**References:**

|  |  |
| --- | --- |
| **SL.No** | **References** |
| **1** | Aurelien Geron, “Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow, OReilly Publications |
| **2** | Francois Chollet, “Deep Learning with Python”, Manning Publications Co, 2nd edition |
| **3** | Introduction to Tensorflow, https:/[/www.tensorflow.org/learn](http://www.tensorflow.org/learn) |
| **4** | Keras Documentation, https://keras.io/ |
| **5** | Ahmed Menshawy, Md. Rezaul Karim, Giancarlo Zaccone, “ Deep Learning with TensorFlow”, Packt Publishing |

# TENSORFLOW & KERAS TUTORIAL

## What is TensorFlow?

TensorFlow is an open-source deep learning framework developed by the Google Brain team. It allows users to create, train, and deploy machine learning models, especially deep neural networks. TensorFlow provides a flexible architecture to work with numerical data using multi-dimensional arrays called **tensors**. It supports both CPU and GPU computations, making it suitable for running on a variety of hardware.

## What are Tensors?

In TensorFlow, tensors are the fundamental data structures used for representing data. They are similar to multi-dimensional arrays and can hold data of any number of dimensions. Tensors are the building blocks of neural networks, as they store the input data, weights, biases, and intermediate outputs during the computation.

Examples of Tensors:

* + 1. Scalar (0-D tensor): A single value is a 0-D tensor. Eg: scalar\_tensor = 5 #rank-0 tensor
    2. Vector (1-D tensor): A 1-D tensor contains a sequence of values.

Eg: vector\_tensor = [1, 2, 3, 4, 5] #rank-1 tensor

* + 1. Matrix (2-D tensor): A 2-D tensor is an array of arrays.

Eg: matrix\_tensor = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] #rank-2 tensor

* + 1. Higher-dimensional tensor (e.g., 3-D tensor):

Eg: tensor\_3d = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]] #rank-3 tensor

Note: For a detailed explanation, visit the TensorFlow | Tensor documentation: <https://www.tensorflow.org/guide/tensor>

## 1. 3 Graph Computation:

TensorFlow follows a symbolic approach for computation using graphs. A graph is a computational graph that represents the flow of data through a series of operations (nodes) to produce output (tensors). The nodes in the graph represent operations, and the edges represent tensors flowing between these operations.

Example of Graph Computation:

import tensorflow as tf

# Define input variables (placeholders) x = tf.placeholder(tf.float32)

y = tf.placeholder(tf.float32)

# Define operations

x\_squared = tf.square(x) # Square operation

x\_squared\_times\_y = tf.multiply(x\_squared, y) # Multiply operation result = tf.add(x\_squared\_times\_y, tf.add(y, 2)) # Add operation

# Create a session to run the computation graph with tf.Session() as sess:

# Provide input values and run the graph

output = sess.run(result, feed\_dict={x: 3.0, y: 4.0}) print("Output:", output)

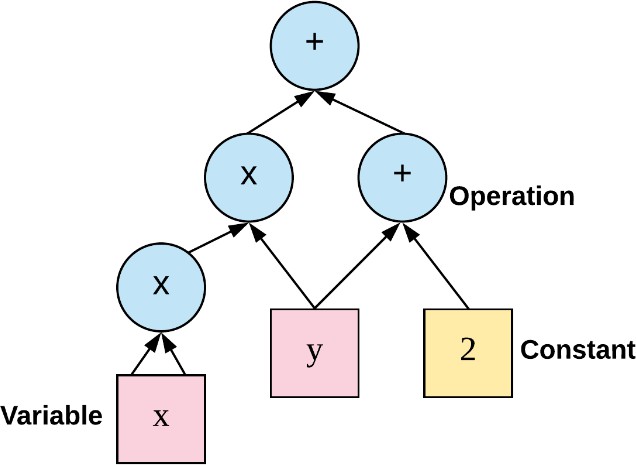


Fig1: Computation graph in tensorflow for **f(x, y) = x2y + y + 2**

[Image Source: https://iq.opengenus.org]

## What is Keras?

Keras is an open-source high-level neural networks API written in Python and capable of running on top of TensorFlow, among other backends. It was designed with a focus on enabling fast experimentation and easy-to-use syntax for building deep learning models. Keras provides a user- friendly interface for constructing complex neural networks, making it an ideal choice for beginners in deep learning.





Fig 2. Tensorflow and Keras as API Image Source: https://developers.google.com/

Note: For a detailed explanation, visit the TensorFlow | Keras documentation: <https://www.tensorflow.org/guide/keras>

In Keras, there are two primary ways to create deep learning models: the **Sequential API** and the

**Functional API**. Each approach serves a different purpose and offers distinct advantages.

## Sequential API:

The Sequential API is the simplest and most straightforward way to build deep learning models in Keras. It allows you to create a linear stack of layers, where each layer has exactly one input tensor and one output tensor. This means that the data flows sequentially through each layer in the order they are added to the model. The Sequential API is well-suited for simple feedforward neural networks and other models that have a clear linear flow of data.

## Example of Sequential API:

from keras.models import **Sequential**

from keras.layers import Dense, Input

# Create a sequential model model = **Sequential()**

# Add layers to the model **model.add**(Input(shape=(input\_dim,))) **model.add**(Dense(64, activation='relu')) **model.add**(Dense(32, activation='relu')) **model.add**(Dense(10, activation='softmax'))

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Print the model summary model.summary()

## Functional API:

The Functional API in Keras allows you to create more complex models with multiple input and output tensors, as well as models with shared layers. It provides greater flexibility and is particularly useful when building models with branching or merging architectures.

Example of Functional API:

from keras.models import Model

from keras.layers import Input, Dense

# Define input tensor

input\_tensor = Input(shape=(input\_dim,))

# Create layers and connect them

hidden\_layer1 = Dense(64, activation='relu')**(input\_tensor)** hidden\_layer2 = Dense(32, activation='relu')**(hidden\_layer1)** output\_tensor = Dense(10, activation='softmax')**(hidden\_layer2)**

# Create the model

model = Model(inputs=input\_tensor, outputs=output\_tensor)

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Print the model summary model.summary()

## Deep Learning Model Life-Cycle

The deep learning model life cycle typically involves the following steps: Define the model, Compile the model, Fit the model, Evaluate the model, and Make predictions.

## Define the Model:

In this step, you specify the architecture of your deep learning model. You define the layers, their configurations, activation functions, and any other required settings. The architecture depends on the problem you are trying to solve, and it may include fully connected layers, convolutional layers, recurrent layers, etc.

from keras.models import Sequential from keras.layers import Dense

# Define the model model = Sequential()

model.add(Dense(64, activation='relu', input\_shape=(input\_dim,))) model.add(Dense(32, activation='relu'))

model.add(Dense(10, activation='softmax'))

## Compile the Model:

After defining the model, you need to compile it. During this step, you specify the loss function, optimizer, and evaluation metrics. The loss function is used to measure how well the model is performing on the training data. The optimizer determines how the model's weights are updated during training, and the evaluation metrics provide additional performance metrics during training.

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

## Fit the Model:

In this step, you train the model on your training data. You provide the input features (X) and their corresponding target labels (y) to the model. The model then adjusts its internal parameters (weights) through an optimization process (usually gradient descent) to minimize the defined loss function.

# Fit the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_val, y\_val))

## Evaluate the Model:

After the model is trained, you need to evaluate its performance on a separate set of data that it has never seen before (e.g., a validation set or a test set). This step gives you an indication of how well the model generalizes to unseen data.

# Evaluate the model

loss, accuracy = model.evaluate(X\_test, y\_test) print(f"Test loss: {loss}, Test accuracy: {accuracy}")

## Make Predictions:

Once the model is trained and evaluated, you can use it to make predictions on new, unseen data. You pass the new data to the model, and it will provide predictions based on what it has learned during training.

# Make predictions

predictions = model.predict(X\_new\_data)

Example: Building a Simple Neural Network with Keras

#1) Import the necessray libraries import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential from tensorflow.keras.layers import Dense, Input

#2) For the tutorial, lets experiment with random data # Generate random input data (features)

X = np.random.rand(num\_samples, num\_features)

# Generate random output labels (classes)

y = np.random.randint(0, num\_classes, size=num\_samples)

# Split the data into training and testing sets split\_ratio = 0.8

split\_index = int(num\_samples \* split\_ratio)

X\_train, X\_test = X[:split\_index], X[split\_index:] y\_train, y\_test = y[:split\_index], y[split\_index:]

#3) Define the model

# Build the neural network model using Sequential API model = Sequential([

Input(shape=(num\_features,)),

Dense(6, activation='relu'), # Hidden layer with 6 neurons Dense(num\_classes, activation='softmax') # Output layer with

num\_classes neurons and softmax activation for classification

])

# Display a summary of the model architecture model.summary()

#4) Compile the model # Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy',

metrics=['accuracy']) #5) Fit/train the model

# Train the model using the training data epochs = 50

batch\_size = 32

model.fit(X\_train, y\_train, epochs=epochs, batch\_size=batch\_size, validation\_split=0.1)

#6) Evaluate/test the model

# Evaluate the model on the testing data

loss, accuracy = model.evaluate(X\_test, y\_test, batch\_size=batch\_size) print("Test Loss:", loss)

print("Test Accuracy:", accuracy)

# WEEK-5: RECURRENT NEURAL NETWORKS FOR TIME SERIES FORECASTING AND STOCK MARK PREDICTION

Q1. Use the following code to generate a time series:

def generate\_time\_series(sample\_size, n\_steps):

freq1, freq2, offsets1, offsets2 = np.random.rand(4, sample\_size, 1) time = np.linspace(0, 1, n\_steps)

series = 0.5 \* np.sin((time - offsets1) \* (freq1 \* 10 + 10)) #wave1+ series += 0.2 \* np.sin((time - offsets2) \* (freq2 \* 20 + 20)) #wave2+ series += 0.1 \* (np.random.rand(sample\_size, n\_steps) - 0.5) #noise return series[..., np.newaxis].astype(np.float32)

The above code generates as many time series as requested, which can be specified using the “sample\_size” argument. Each time series is of length “n\_steps” and there is just one value per time step in each series.

Use the above code to do the following:

* 1. Create a dataset of 10,000 samples with 51-time steps each (Note: the 51st time step should be used as the label)
  2. Split the dataset in the ratio training: validation: testing = 70:20:10.
  3. Design, train, test and compare the performances of the following on the prediction of the value of 51st time step in the generated time series.
     1. Fully connected neural network.
     2. Simple RNN with one hidden layer and one output layer.
     3. Simple RNN with two hidden layers and one output layer.

Q2. Consider the **Google Stock Prediction dataset.** The 14 columns in the dataset are as follows:

|  |  |
| --- | --- |
| * symbol : - Name of the company (in this case Google). | * adjClose |
| * date :- year and date | * adjHigh |
| * close:- closing of stock value | * adjLow |
| * high:- highest value of stock at that day | * adjOpen |
| * low:- lowest value of stock at that day | * adjVolume |
| * open:- opening value of stock at that day | * divCash |
| * volume | * splitFactor |

1. Build a Simple RNN model with 2 hidden layers and 1 Dense layer to predict the stock price for the years 2020 and 2021.
2. Compare the accuracy using MAPE and MSE.
3. Comment on how many epochs (dropouts) are required for adequate learning.
4. Plot the actual vs predicted values using the test data for the year 2020 and 2021.

Q3. **Burglary Dataset:** Baltimore is a city significantly known for high crime rate which ranks higher than the national average. Each crime record comes with both spatial (latitude and longitude) and temporal (date and time of occurrence) information along with the specific type of crime. This includes eleven different categories of crimes such as homicide, robbery, larceny etc.

Consider the crimes of LARCENY (Crime code starting with 6), BURGLARY (Crime code starting with 5). Create two time series datasets LarcenyTs, BurglaryTs to represent the total number of crimes, day-wise. Put data from 2014, 2015 into training and predict the total number of LARCENY and BURGLARY crimes for the year 2016.

1. Build a **Simple RNN model vs a LSTM model,** both with 4 layers to predict the total number of LARCENY and BURGLARY crimes for the year 2016.
2. Compare and comment on their accuracy using MAPE, RMSE.
3. Comment on how many epochs are required for adequate learning.
4. Plot the actual vs predicted values using the test data for the year 2016.