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

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

LABORATORY MANUAL

***Instructors*:**

Dr. Rohini R Rao,

Dr. Abhilash K Pai,

Dr. Saraswati Koppad

Department of Data Science & Computer Applications

Manipal Institute of Technology, Manipal, India

JULY 2024

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.	9	15
CO2	Design, develop, fine-tune, evaluate simple and advanced CNN models for Image classification.	10	35
CO3	Design, develop, fine-tune, evaluate simple and advanced RNN models for sequence modelling tasks like Time series prediction and NLP.	11	35
CO4	Design, develop, fine-tune, and evaluate Autoencoders and Generative models for representational learning.	6	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: ○ Record : 6M, ○ Program execution : 7M, ○ Quiz : 7M • 1 Mid-Sem Examination: 20 marks • Mini Project : 20 marks ○ Phase1: Problem + Literature: 5M ○ Phase 2: End-to-End solution: 8M ○ 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 Phase 1 Evaluation	CO2
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, Image Generation Using Generative Adversarial Networks	CO4
Week 10	Fine tuning of LLM for NLP tasks	CO4
Week 11	Mini Project Final Evaluation	CO1
Week 12	End-term lab examination	

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
4	Keras Documentation, https://keras.io/
5	Ahmed Menshawy, Md. Rezaul Karim, Giancarlo Zaccone, " Deep Learning with TensorFlow", Packt Publishing

TENSORFLOW & KERAS TUTORIAL

1.1 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.

1.2 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`

3. 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`

4. 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)
```

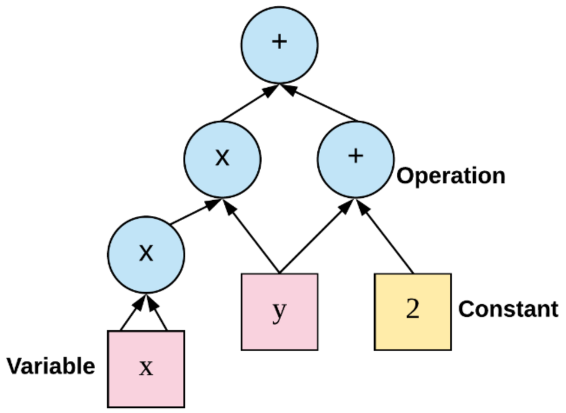

Fig1: Computation graph in tensorflow for **f(x, y) = x^2y + y + 2**
[Image Source: https://iq.opengenus.org]

1.4 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.

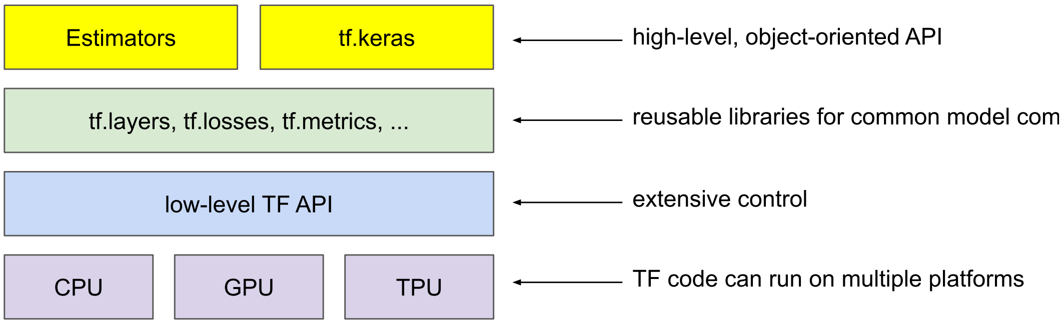

Fig 2. Tensorflow and Keras as API
Image Source: https://developers.google.com/

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.

1.5 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()
```

1.6 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()
```

1.7 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.

I. 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'))
```

II. 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'])
```

III. 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))
```

IV. 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}")
```

V. 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
```

```python
# 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-1: GETTING STARTED WITH BUILDING FULLY CONNECTED NEURAL NETWORKS IN KERAS

Q1. Using the Iris Flowers Dataset, build and Neural Network with the following specifications to perform multi-class classification.

- Split the Data into Training: Validation: Testing
- Number of Hidden Layers = 2, containing 8 Neurons and 4 Nuerons
- Use RELU activation function in the hidden layers, choose the optimizer as ADAM and set learning rate to be equal to 0.1.

Q2. Accurate measurement of body fat is inconvenient/costly, and it is desirable to have easy methods of predicting Body Fat. Using the given Body Fat dataset, build a Neural Network to predict body fat. Plot the training and validation performance curves and analyze the performance of the proposed neural network.

The attributes of the dataset are as follows:

1. Density determined from underwater weighing
2. Percent body fat from Siri's (1956) equation
3. Age (years)
4. Weight (lbs)
5. Height (inches)
6. Neck circumference (cm)
7. Chest circumference (cm)
8. Abdomen 2 circumference (cm)
9. Hip circumference (cm)
10. Thigh circumference (cm)
11. Knee circumference (cm)
12. Ankle circumference (cm)
13. Biceps (extended) circumference (cm)
14. Forearm circumference (cm)
15. Wrist circumference (cm)

Use the following hyperparameters/design choices for your neural network:

- Split the data in the ratio Training: Validation: Testing = 80:10:10.
- Perform Normalization using Standard Scalar.
- Number of Hidden layers = 3 and number of units for each hidden layers are 128,64,32, respectively.
- Use RELU activation function in the hidden layers, choose the optimizer as ADAM and set learning rate to be equal to 0.1.

Q3. For Q1 and Q2, Interpret the results of “model.summary()” (use comments/markup in the notebook)