Experiment No. 1	
BE (AI&DS)	ROLL NO : 9742
Date of Implementation: 02/08/2024	
Aim: Study of Deep Learning Packages: TensorFlow, Keras, Theano and PyTorch. Document the distinct features and functionality of the packages	
Programming Language Used :PYTHON	
Upon completion of this experiment, students will be able to LO3 : Build and train deep learning models for given problem.	

Indicator		
Timeline Maintains submission deadline (1)	On time (1)	Otherwise (0)
Completion and Organization (2)	Completed in LAB (2)	Otherwise(1)
Analysis of output and conclusion(2)	Properly done (2)	Otherwise (0)
Viva (10)		

Assessment Marks :

Timeline(1)	
Completion and Organization (2)	
Analysis of output and conclusion(2)	
Viva (10)	
Total (15)	

EXPERIMENT	***1***
Aim	Study of Deep Learning Packages: TensorFlow, Keras, Theano and PyTorch. Document the distinct features and functionality of the packages
Tools	Python
Theory	What is Deep Learning? Deep learning can be considered as a subset of machine learning. It is a field that is based on learning and improving on its own by examining computer algorithms. Until recently, neural networks were limited by computing power and thus were limited in complexity. However, advancements in Big Data analytics have permitted larger, sophisticated neural networks, allowing computers to observe, learn, and react to complex situations faster than humans. Artificial neural networks, comprising many layers, drive deep learning. Deep Neural Networks (DNNs) are such types of networks where each layer can perform complex operations such as representation and abstraction that make sense of images, sound, and text. Python libraries that are used in Machine Learning are: Numpy Scipy Scikit-learn Theano TensorFlow Keras PyTorch Pandas Matplotlib
Implementation	1. Install required packages 2. Import packages and print the version 3. Compare Tensorflow, Keras, Theano and PyTorch
Conclusion	In this experiment, We studied different deep learning packages like TensorFlow, Keras, PyTorch and Theano by installing them. We Imported the packages and used them in the code to understand the basics of these packages and also printed the version. Later, the comparison between all these packages are made.

<u>Implementation:</u>

1). Install required packages.

```
pip install tensorflow
pip install keras
pip install theano
pip install torch
```

2). Import packages and print the version.

<u>Tensorflow:</u>

Code:

```
import tensorflow as tf
import numpy as np

print("TensorFlow version:", tf.__version__)

# Generate some random data
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 * X + 1 + np.random.randn(100, 1) * 0.1

# Define a simple model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(1, input_shape=(1,))
])

# Compile the model
model.compile(optimizer='sgd', loss='mse')

# Train the model
model.fit(X, y, epochs=100, verbose=0)

# Make predictions
X_test = np.array([[0.5]])
prediction = model.predict(X_test)

print(f"Prediction for input 0.5: {prediction[0][0]:.4f}")
```

Output:

```
TensorFlow version: 2.17.0
Prediction for input 0.5: 2.0280
```

<u>Keras:</u>

Code:

```
import keras
from keras.models import Sequential
from keras.layers import Dense
import numpy as np

print("Keras version:", keras.__version__)

# Generate some random data
np.random.seed(0)
X = np.random.randn(1000, 2)
y = (X[:, 0] + X[:, 1] > 0).astype(int)
```

```
# Create a model
model = Sequential([
    Dense(4, activation='relu', input_shape=(2,)),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X, y, epochs=50, validation_split=0.2, verbose=0)

# Evaluate the model
loss, accuracy = model.evaluate(X, y)
print(f"Accuracy: {accuracy:.4f}")
```

Output:

Keras version: 3.4.1
Accuracy: 0.9880

PyTorch:

Code:

```
import torch
import torch.nn as nn
import numpy as np

print("PyTorch version:", torch.__version__)

# Generate some random data
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 * X + 1 + np.random.randn(100, 1) * 0.1

# Convert NumPy arrays to PyTorch tensors
X_tensor = torch.from_numpy(X).float()
y_tensor = torch.from_numpy(y).float()

# Define a simple model
class LinearRegression(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(1, 1)

    def forward(self, x):
        return self.linear(x)
```

```
model = LinearRegression()

# Define loss and optimizer
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)

# Train the model
for epoch in range(100):
    # Forward pass
    y_pred = model(X_tensor)
    loss = criterion(y_pred, y_tensor)

    # Backward pass and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

# Make a prediction
X_test = torch.tensor([[0.5]])
with torch.no_grad():
    prediction = model(X_test)

print(f"Prediction for input 0.5: {prediction.item():.4f}")
```

Output:

```
PyTorch version: 2.3.1+cu121
Prediction for input 0.5: 1.7918
```

Theano:

Code:

```
import theano
import theano.tensor as T
 import numpy as np

print("Theano version:", theano.__version__)

# Create some random input data using NumPy
A = np.random.rand(3, 4).astype(theano.config.floatX)
B = np.random.rand(4, 2).astype(theano.config.floatX)

# Define the symbolic variables
a = T.matrix('a')
b = T.matrix('b')
```

```
# Define the operation
c = T.dot(a, b)

# Compile the Theano function
func = theano.function(inputs=[a, b], outputs=c)

 # Perform the matrix multiplication

 result = func(A, B)

print("Matrix A:")
print(A)
print("\nMatrix B:")
print(B)
print("\nResult:")
print(result)
```

Output:

```
Theano version: 1.0.5

Matrix A:
[[0.4983524  0.34774273 0.3178267  0.68109953]
 [0.02974079 0.8437361  0.95449805 0.4137837 ]
 [0.9290739  0.5282176  0.0940957  0.08868537]]

Matrix B:
[[0.04908372 0.03994029]
 [0.15885887 0.3680199 ]
 [0.15726647 0.9586587 ]
 [0.8050547  0.8832174 ]]

Result:
[[0.5798819  0.8594063 ]
 [0.9782976  1.4747598 ]
 [0.5853889  1.2254114 ]]
```

3). Compare Tensorflow, Keras, Theano and PyTorch.

Parameters	Tensorflow	Keras	Theano	PyTorch
Architecture	Static computation graph	High-level API on top of TensorFlow, Theano, or CNTK	Static computation graph	Dynamic computation graph
Flexibility	Extensive control over low-level operations	Emphasizes ease of use and rapid prototyping	Provides control but requires more effort	Focuses on control and flexibility for custom models
Model Building	Strong deployment capabilities, production-ready	Facilitates rapid prototyping and experimentation	Detailed control over computations	Quick iterations, detailed debugging
Speed and Efficiency	Optimized for large-scale models, high performance	Performance depends on the backend (TensorFlow, Theano, CNTK)	Good performance, but less optimized than TensorFlow	Efficient for small to medium-scale models
Scalability	Highly scalable, handles enterprise-level deployments	Scales well for high-level applications via TensorFlow	Scales well but less frequently used in production	Suitable for research, can scale with effort
Popularity	Widely adopted in industry and academia	Widely adopted for its simplicity and ease of use	Historically important, now less commonly used	Gaining traction in academia and research
Community and Support	Large community, strong Google support	Extensive documentation, strong community backing from TensorFlow	Smaller community compared to TensorFlow and PyTorch	Growing community, strong support from Facebook