Name: Nupur Lalit Vartak
Class: BE/CSE(DS)
Roll No.: 62
Subject: Deep Learning Lab
Exp No.: 02

Aim: To explore python libraries for deep learning e.g. Keras, TensorFlow etc.

Keras

Keras is an open-source neural network library written in Python. It is designed to be user-friendly, modular, and extensible, allowing for rapid experimentation with deep learning models. Keras supports both convolutional and recurrent networks, and can run on top of TensorFlow, Theano, or CNTK.

Import Keras:

Importing Keras includes the entire library, while `from keras.models import Sequential` imports the `Sequential` class for creating layer stacks, and `from keras.layers import Dense, Conv2D, LSTM` imports specific layer types (Dense, Conv2D, LSTM) for neural network architecture definition.

```
# Importing the entire keras module
import keras

# Importing specific components from keras
from keras.models import Sequential
from keras.layers import Dense, Conv2D, LSTM
```

Create a Sequential model:

Use the Sequential class to create a linear stack of layers. This is the simplest way to build a model in Keras.

```
model = Sequential()
```

Add layers:

Add layers to the model using the add() method. Specify the type of layer you want to add (e.g., Dense, Conv2D, etc.), along with the desired number of units/neurons and activation function.

```
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))
```

Compile the model:

Compile the model using the compile() method. Specify the loss function, optimizer, and metrics to be used during training.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Train the model:

Train the model using the fit() method. Provide the training data (X_train, y_train), specify the batch size, number of epochs, and any additional parameters required.

```
X_train = [[0] * 100 for _ in range(100)]
y_train = [[0] * 10 for _ in range(100)]

model.fit(X_train, y_train, batch_size=32, epochs=10)
```

Evaluate the model:

Evaluate the trained model on test data using the evaluate() method. Provide the test data (X_test, y_test) and observe the performance metrics.

```
X_test = [[0] * 100 for _ in range(10)]
y_test = [[0] * 10 for _ in range(10)]

loss, accuracy = model.evaluate(X_test, y_test)
print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

Make predictions:

Use the trained model to make predictions on new/unseen data using the predict() method. Provide the input data and obtain the model's predictions.

```
X_new = [[0] * 100 for _ in range(5)]

predictions = model.predict(X_new)
```

Model Summary:

Display a summary of the model architecture, showing layer types, output shapes, and number of parameters.

```
model.summary()
```

TensorFlow

TensorFlow is an open-source machine learning framework developed by Google. It allows developers to build and deploy machine learning models efficiently across a variety of platforms, from mobile devices to large-scale distributed systems. TensorFlow provides a comprehensive ecosystem of tools, libraries, and community resources to support deep learning and other machine learning tasks.

The word TensorFlow is made by two words, i.e., Tensor and Flow

1. Tensor is a multidimensional array
2. Flow is used to define the flow of data in operation.

Import TensorFlow:

Import TensorFlow as `tf`, enabling access to its modules, classes, functions, and operations; ensure TensorFlow is installed (`pip install tensorflow`) beforehand.

```
#Import Tensorflow library
import tensorflow as tf
```

tf.constant:

Creates a constant tensor with specified value(s). This operation is fundamental for defining tensors with fixed values that do not change during computation.

```
tensor = tf.constant(5.0)
# Creates a constant tensor with value 5.0
```

tf.Variable:

Creates a variable tensor that can be updated during training. Variables are mutable tensors used to represent model parameters that are updated during training using gradient descent or its variants.

```
variable = tf.Variable(initial_value=[1.0, 2.0, 3.0])
# Creates a variable tensor with initial values [1.0, 2.0, 3.0]
```

tf.keras.layers.Dense:

Dense (fully connected) layer for neural networks. This method creates a fully connected layer where each neuron is connected to every neuron in the preceding layer, suitable for learning complex patterns in data.

```
layer = tf.keras.layers.Dense(units=64, activation='relu')
# Creates a dense layer with 64 units and ReLU activation function
```

tf.reduce_mean:

Computes the mean of elements across dimensions of a tensor. Used for reducing tensor dimensions by computing the average value, which is useful for aggregating metrics like loss or accuracy.

```
mean = tf.reduce_mean([1, 2, 3, 4, 5])
# Computes the mean of [1, 2, 3, 4, 5]
```

tf.GradientTape:

Records operations for automatic differentiation and gradient computation. Enables automatic differentiation in TensorFlow, crucial for computing gradients of tensors with respect to other tensors.

```
with tf.GradientTape() as tape:
    x = tf.Variable(3.0)
    y = x * x
dy_dx = tape.gradient(y, x)
```

tf.keras.optimizers.Adam:

Adam optimizer for gradient-based optimization. Adam is an adaptive learning rate optimization algorithm that combines the advantages of AdaGrad and RMSProp, suitable for a wide range of optimization tasks in deep learning.

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
# Creates an Adam optimizer with a learning rate of 0.001
```

tf.data.Dataset:

API for building input pipelines with data transformation and batching. Simplifies the process of creating data input pipelines by enabling efficient data loading, transformation, and batching for model training.

```
import numpy as np
import tensorflow as tf

features = np.array([[1, 2], [3, 4], [5, 6]])
labels = np.array([0, 1, 0])

dataset = tf.data.Dataset.from_tensor_slices((features, labels)).batch(32)
# Creates a dataset from tensors and batches data in size 32
```

tf.keras.Model.compile:

Compiles a Keras model with specified loss, optimizer, and metrics. Configures the model for training by specifying the optimizer, loss function, and evaluation metrics, preparing it for the training and evaluation phases.

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Conclusion:

Exploring Python libraries for deep learning, such as Keras and TensorFlow, opens up a wealth of powerful tools for developing sophisticated machine learning models. These libraries provide intuitive APIs, extensive documentation, and strong community support, making them ideal for both beginners and seasoned practitioners to tackle complex AI challenges effectively and efficiently. Whether building neural networks, optimizing models, or deploying solutions, Keras and TensorFlow offer versatile frameworks that continue to drive innovation in the field of deep learning.