EXPERIMENT - 1

AIM: Setting up the Spyder IDE Environment and Executing a Python Program
THEORY:

Spyder IDE

Spyder (Scientific Python Development Environment) is an open-source IDE designed mainly for scientific programming in Python. It comes with Anaconda distribution, making it popular for data science and experimentation.
Features:

- **Integrated Environment**: Provides an editor, console, variable explorer, and debugging tools in one place.
- **Editor**: Allows writing and saving .py files with syntax highlighting and auto-completion.
- **Console**: Lets you run code interactively or execute complete Python scripts.
- **Variable Explorer**: Shows variables, arrays, and data in a tabular format.
- **Ease of Setup**: Comes bundled with Anaconda, so you don't need separate package installations.

Google Colab

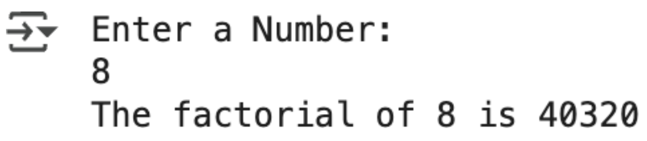
Google Colaboratory (Colab) is a cloud-based platform by Google for writing and running Python code in a web browser.
Features relevant to this experiment

- **No installation needed**: Runs completely online, only requires a Google account.
- **Notebook Interface**: Uses Jupyter Notebook style with cells for code and text.
- **Cloud Execution**: Code runs on Google's servers, not your local machine.
- **Preinstalled Libraries**: Popular packages like NumPy, pandas, TensorFlow are ready to use.
- **Hardware Options**: Can leverage free GPUs/TPUs for computation-heavy tasks.
- **Collaboration**: Easy to share and work with others in real time.

CODE:

```
# Program to find factorial of a number
def factorial(n):
    if n == 0 | n==1:
        return 1
    else:
        return n * factorial(n-1)

print("Enter a Number: ")
num = int(input())
print("The factorial of", num, "is", factorial(num))
```

OUTPUT:



```
Enter a Number:
8
The factorial of 8 is 40320
```

LEARNING OUTCOME:

EXPERIMENT - 2

AIM: Installing Keras, Tensorflow and Pytorch libraries and making use of them

THEORY:

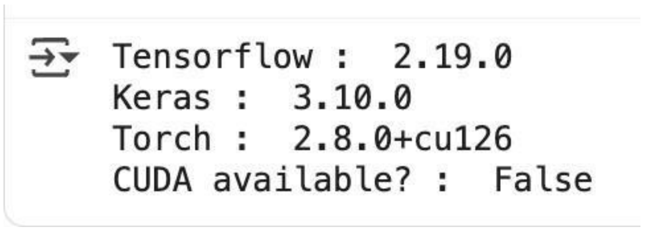
TensorFlow is an open-source deep learning framework developed by Google. It is designed for handling large-scale machine learning tasks and supports both CPU and GPU computation. TensorFlow is highly popular in industry because it offers scalability, performance, and strong deployment capabilities across platforms such as cloud, web, and mobile. Its rich ecosystem provides tools for preprocessing, building, training, and deploying models, making it a reliable choice for production-level applications.

Keras is a high-level neural network API that is now integrated with TensorFlow but can also function independently. It focuses on simplicity and ease of use, allowing quick experimentation and rapid prototyping of deep learning models. Keras hides much of the complexity of lower-level frameworks, which makes it beginner-friendly and ideal for learning, teaching, and smaller projects. While it may not offer the same level of control as TensorFlow or PyTorch, it greatly speeds up model development.

PyTorch, created by Facebook (Meta), is another widely used deep learning framework. It is known for its dynamic computation graphs, which allow flexibility and easier debugging compared to static graph frameworks. PyTorch has become the framework of choice in research because of its intuitive, Python-like design and seamless integration with scientific libraries. Though it started as a research tool, PyTorch is now increasingly adopted in industry as well, especially in areas like computer vision and natural language processing where experimentation and flexibility are important.

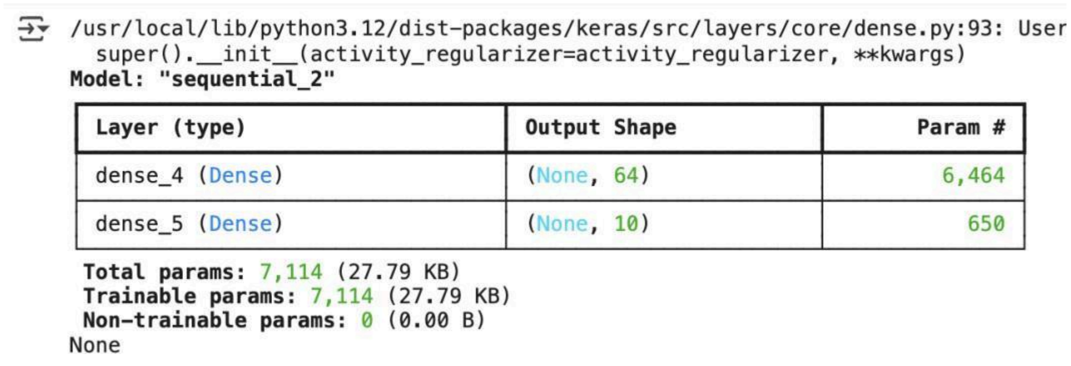
CODE:

```
# SIMPLE NEURAL NETWORK

import tensorflow as tf # has keras in
it import torch

print("Tensorflow : ",tf._version_)
print("Keras : ",tf.keras._version_)
print("Torch : ",torch._version_)
print("CUDA available? : ",torch.cuda.is_available())
```



```
Tensorflow :  2.19.0
Keras :  3.10.0
Torch :  2.8.0+cu126
CUDA available? :  False
```

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

model = tf.keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(100,)), # for
    input layers.Dense(10, activation='softmax'), # for output

])

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

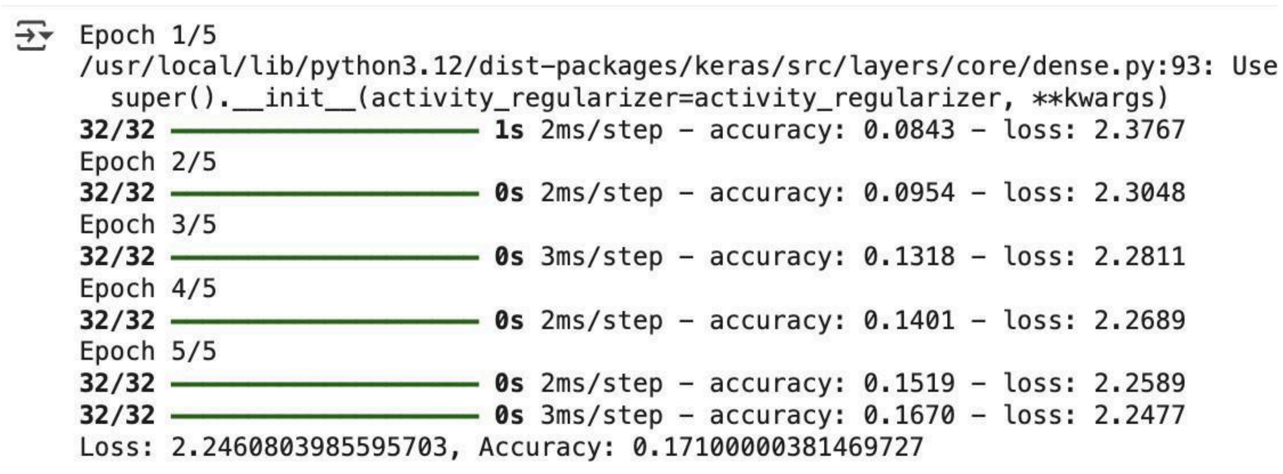


```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: User
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_2"
```

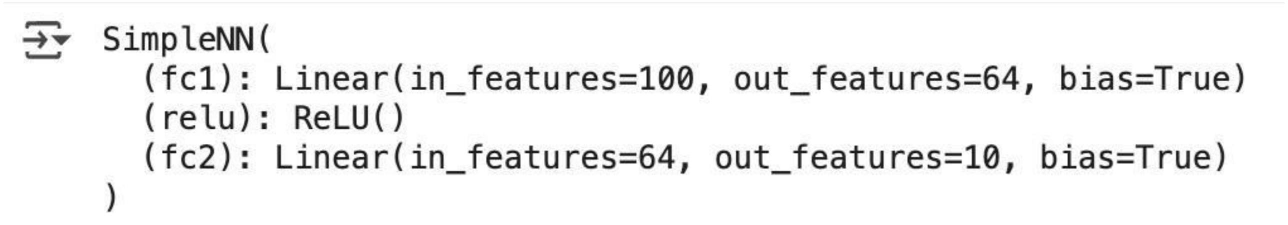
Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 64)	6,464
dense_5 (Dense)	(None, 10)	650

```
 Total params: 7,114 (27.79 KB)
 Trainable params: 7,114 (27.79 KB)
 Non-trainable params: 0 (0.00 B)
None
```

```
import torch
import torch.nn as nn #
import torch.optim as
optim

#define simple network
class SimpleNN(nn.Module):
  def _init_(self):
    super(SimpleNN, self)._init_()
    self.fc1=nn.Linear(100,64)
    self.relu=nn.ReLU()
    self.fc2=nn.Linear(64,10)

  def forward(self,x):
    x =
    self.relu(self.fc1(x)) x
    = self.fc2(x)
    return x

#create model, loss and optimizer
model=SimpleNN()
criterion=nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

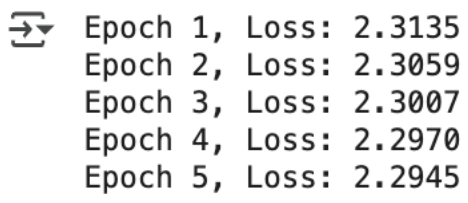


```
Epoch 1/5
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: Use
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.0843 - loss: 2.3767
Epoch 2/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.0954 - loss: 2.3048
Epoch 3/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.1318 - loss: 2.2811
Epoch 4/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1401 - loss: 2.2689
Epoch 5/5
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1519 - loss: 2.2589
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.1670 - loss: 2.2477
Loss: 2.2460803985595703, Accuracy: 0.17100000381469727
```



```
SimpleNN(
  (fc1): Linear(in_features=100, out_features=64, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=64, out_features=10, bias=True)
)
```

```
print(model)
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import
layers import numpy as np

# Dummy data (1000 samples, 100 features each)
X_train = np.random.rand(1000, 100).astype(np.float32)
y_train = np.random.randint(0,10, size=1000) # 10 classes

# Define model
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(100,)),
    layers.Dense(10, activation='softmax')
])

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

# Train
history = model.fit(X_train, y_train, epochs=5, batch_size=32)

# Evaluate
loss, accuracy = model.evaluate(X_train, y_train)
print(f"Loss: {loss}, Accuracy: {accuracy}")


import torch
import torch.nn as nn
import torch.optim as optim

# Dummy dataset
X_train = torch.rand(1000, 100) # 1000 samples, 100 features
y_train = torch.randint(0, 10, (1000,)) # 10 classes

# Define simple network
class SimpleNN(nn.Module):
    def _init_(self):
        super(SimpleNN, self)._init_()
        self.fc1 = nn.Linear(100, 64)
        self.fc2 = nn.Linear(64, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x =
        self.relu(self.fc1(x)) x
        = self.fc2(x)
        return x

# Create model, loss, optimizer
model = SimpleNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Train
for epoch in range(5):
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()
                                           print(f"Epoch {epoch+1}, Loss:
                                       {loss.item():.4f}")
```



```
Epoch 1, Loss: 2.3135
Epoch 2, Loss: 2.3059
Epoch 3, Loss: 2.3007
Epoch 4, Loss: 2.2970
Epoch 5, Loss: 2.2945
```

LEARNING OUTCOME: