TensorFlow

TensorFlow is a free and open-source machine learning framework developed by Google mainly used to build, train and deploy machine learning and deep learning models. It supports numerous tasks such as image recognition and natural language processing. It is available on both CPU, GPU and TPU without hiccups and the easy-to-use in Keras API.

Import Tensor-Flow

```
import tensorflow as tf
```

Basic Operations

Command	Execution
a = tf.constant(5) b = tf.constant(3) c = a + b print(c.numpy())	In TensorFlow, a constant is an immutable tensor that is meant to store values fixed throughout the runtime of a program.

x = tf.Variable(10)
x.assign(15) # Update value
print(x.numpy())
	In TensorFlow, a variable is an object representing a shared persistent state modified during program execution.

Tensors

Command	Execution
tensor = tf.constant([[1, 2], [3, 4]]) print(tensor)	In TensorFlow, you can construct tensors using various functions such as `tf.constant()` for constant value, `tf.zeros()` to fill a tensor with zeros and `tf.ones()` to fill the tensor with ones.
reshaped = tf.reshape(tensor, [4, 1]) print(reshaped)	Reshaping tensors in TensorFlow changes the shape of a tensor without changing its data.

Optimizers

Command	Execution
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)	Optimizers are a must-have in any TensorFlow application for adjusting the weights in a model towards minimizing the loss function during the training process.

Loss Functions

Command	Execution
loss = tf.keras.losses.MeanSquaredError()	Loss functions are one of the most important things when training a machine learning model in TensorFlow as they represent the difference between the actual target values and the predicted values

Training and Evaluation

Command	Execution
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])	Compiling a model in TensorFlow is one of the essential steps that have to be performed before training and evaluating the model.
model.fit(x_train, y_train, epochs=10, batch_size=32)	Train a model in TensorFlow by importing necessary libraries, loading your dataset, preprocessing it (e.g., normalize pixel values) defining the architecture of your neural network using the Keras API.
model.evaluate(x_test, y_test)	To evaluate a model from TensorFlow, you would use the `model.evaluate()` method, which evaluates the model's performance on the dataset by computing the loss and optionally selected metrics.

TensorFlow Datasets

Command	Execution
mnist = tf.keras.datasets.mnist (x_train, y_train), (x_test, y_test) = mnist.load_data() x_train, x_test = x_train / 255.0, x_test / 255.0	we can use the function `tfds.load()` from the TensorFlow Datasets (TFDS) library to load a dataset in TensorFlow.

Saving and Loading Models

Command	Execution
model.save('model_name.h5')	You can save a model in TensorFlow with the model.save() method, which saves the entire architecture, weights, and optimizer state of the model.
new_model = tf.keras.models.load_model('model_name.h5')	To load a model in TensorFlow, you might use the following function called tf.keras.models.load_model() which allows you to read a saved model from the storage for further usage.

GPU Utilization

Command	Execution
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))	checking GPU availability in TensorFlow
gpus = tf.config.experimental.list_physical_devices('GPU') if gpus: tf.config.experimental.set_memory_growth(gpus[0], True)	This code snippet is used to enable the growth of GPU memory in TensorFlow, which allocates GPU memory incrementally as needed instead of preallocating all available memory.

TensorFlow Utilities

Command	Execution
with tf.GradientTape() as tape: predictions = model(x_train) loss_value = loss(y_train, predictions) grads = tape.gradient(loss_value, model.trainable_variables) optimizer.apply_gradients(zip(grads, model.trainable_variables))	tf.GradientTape in TensorFlow enables automatic differentiation, making it ideal for implementing custom training loops
numpy_array = tensor.numpy()	Converting a TensorFlow Tensor into a NumPy Array A NumPy array from a TensorFlow tensor can be acquired through the tensor object using its `numpy()` method.

TensorFlow Lite

Command	Execution
converter = tf.lite.TFLiteConverter.from_saved_model('model_name') tflite_model = converter.convert()	To convert a TensorFlow model into TFLite, you would use the class `tf.lite.TFLiteConverter`.

Import TensorFlow

```
import tensorflow as tf
print(tf.__version__)
```

Define Tensors

```
a = tf.constant(5)
b = tf.constant(3)
result = a + b
print("AdditionResult:", result.numpy())
```

Addition Result: 8

Creating Tensor

```
tensor = tf.constant([[1, 2], [3, 4]])
print("Tensor:\n", tensor.numpy())
```

Tensor:
[[1 2]
[3 4]]

Perform Matrix Operations

```
matrix1 = tf.constant([[1, 2], [3, 4]])
matrix2 = tf.constant([[2, 0], [1, 2]])
result = tf.matmul(matrix1, matrix2)
print("Matrix Multiplication Result:\n", result.numpy())
```

Output

Matrix Multiplication Result:
[[4 4]
[10 8]]

Build Neural Network

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

# Create a model
model = Sequential([
  Dense(10, activation='relu', input_shape=(2,)),
  Dense(1, activation='sigmoid')
])

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

# Display the model summary
model.summary()
```

Output

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	30
dense_1 (Dense)	(None, 1)	11

Total params: 41

Trainable params: 41

Non-trainable params: 0

Training Model

```
import numpy as np

# Generate some dummy data
X = np.random.rand(100, 2)
y = np.random.randint(0, 2, size=(100,))

# Train the model
model.fit(X, y, epochs=5, batch_size=10)
```

Output

```
Epoch 1/5
10/10 - 0s 1ms/step - loss: 0.6951 - accuracy:
0.5100
Epoch 2/5
10/10 - 0s 1ms/step - loss: 0.6940 - accuracy:
0.5200
```

Making Model Prediction

```
# Predict on new data
test_data = np.random.rand(5, 2)
predictions = model.predict(test_data)
print("Predictions:\n", predictions)
```

Output

```
Predictions:
[[0.54983985]
 [0.50234926]
 [0.49029356]
 [0.52347356]
 [0.56718266]]
```

Save and Load Model

```
# Save the model
model.save('my_model.h5')

# Load the model
loaded_model= tf.keras.models.load_model('my_model.h5')
print("Model loaded successfully!")
```

Output

Model loaded successfully!

Use TensorFlow for customer Training Loops

```
# Define a custom training loop
optimizer = tf.keras.optimizers.Adam()
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()

@tf.function
def train_step(x, y):
  with tf.GradientTape() as tape:
    predictions = model(x, training=True)
    loss = loss_fn(y, predictions)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))
  return loss

# Example loop
for epoch in range(3):
  for i in range(100):  # Assuming batch size 100
    loss = train_step(x_train[i:i+100], y_train[i:i+100])
  print(f"Epoch {epoch + 1}, Loss: {loss.numpy()}")
```

Output

```
Epoch 1, Loss: 2.3095782
Epoch 2, Loss: 2.308771
Epoch 3, Loss: 2.3081365
```

GeeksforGeeks