Pavan Kumar Adinani

Tensor Basics

What's a Tensor?

A tensor is a fundamental data structure in TensorFlow, similar to NumPy arrays but with additional capabilities for computation across GPUs and TPUs. It's a multi-dimensional array used for numerical computation in deep learning.

- Tensors can be scalars (0D), vectors (1D), matrices (2D), or higher-dimensional structure (3D, 4D, etc.)
- Tensors support GPU acceleration, making computation faster.

Tensor Type	Example Representation	Shape
0D Tensor (Scalar)	6	( ) (single value)
1D Tensor (Vector)	[1, 2, 3]	(3,)
2D Tensor (Matrix)	[[1, 2, 3], [4, 5, 6]]	(2, 3) (rows, columns)
3D Tensor (Cube)	[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]	(2,, 2, 2)
4D Tensor (Batch of Images)	(32, 64, 64, 3) for images (Batch, height, width, channels)	(32, 64, 64, 3)

Code Snippets:

```python
import tensorflow as tf
import numpy as np

def print_tensor(name, tensor):
    print(f"{name}:")
    print(tensor.numpy())
    print(f"Shape: {tensor.shape}\n{'-'*40}")

# 0D Tensor (Scalar)
scalar = tf.constant(6)
print_tensor("0D Tensor (Scalar)", scalar)

# 1D Tensor (Vector)
vector = tf.constant([1, 2, 3])
print_tensor("1D Tensor (Vector)", vector)

# 2D Tensor (Matrix)
matrix = tf.constant([[1, 2, 3], [4, 5, 6]])
print_tensor("2D Tensor (Matrix)", matrix)

# 3D Tensor (Cube)
cube = tf.constant([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print_tensor("3D Tensor (Cube)", cube)

# 4D Tensor (Batch of Images)
batch_images = tf.random.uniform(shape=(32, 64, 64, 3))
print(f"4D Tensor (Batch of Images) Shape: {batch_images.shape}")
```

Output

```
0D Tensor (Scalar):
6
Shape: ()
----------------------------------------
1D Tensor (Vector):
[1 2 3]
Shape: (3,)
----------------------------------------
2D Tensor (Matrix):
[[1 2 3]
 [4 5 6]]
Shape: (2, 3)
----------------------------------------
3D Tensor (Cube):
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
Shape: (2, 2, 2)
----------------------------------------
4D Tensor (Batch of Images) Shape: (32, 64, 64, 3)
```

Tensor Data Types & Shapes in TensorFlow

TensorFlow supports various data types, each optimised for different kinds of computations.

Here are some commonly used ones:

Data Type	Description	Example
tf.float32	32-bit floating-point numbers(default for most operations)	`tf.constant(3.14, dtype=tf.float32)`
tf.float64	64-bit floating-point numbers (higher precision)	`tf.constant(3.14159, dtype=tf.float64)`
tf.int32	32-bit integer	`tf.constant(42, dtype=tf.int32)`
tf.int64	64-bit integer	`tf.constant(123456789, dtype=tf.int64)`
tf.uint8	Unsigned 8-bit integer (used for images, e.g., 0-255 pixel values)	`tf.constant(255, dtype=tf.uint8)`
tf.bool	Boolean values (True/False)	`tf.constant(True, dtype=tf.bool)`
tf.string	String tensors	`tf.constant(“Hello, TensorFlow!”`

Code Snippets:

```python
import tensorflow as tf

# Creating tensors with different data types
tensor_float = tf.constant(3.14, dtype=tf.float32)
tensor_int = tf.constant(42, dtype=tf.int32)
tensor_bool = tf.constant(True, dtype=tf.bool)
tensor_string = tf.constant("Hello, TensorFlow!", dtype=tf.string)

# Checking tensor properties
print(f"Tensor: {tensor_float}, Shape: {tensor_float.shape}, Type: {tensor_float.dtype}")
print(f"Tensor: {tensor_int}, Shape: {tensor_int.shape}, Type: {tensor_int.dtype}")
print(f"Tensor: {tensor_bool}, Shape: {tensor_bool.shape}, Type: {tensor_bool.dtype}")
print(f"Tensor: {tensor_string}, Shape: {tensor_string.shape}, Type: {tensor_string.dtype}")
```

```
Tensor: 3.140000104904175, Shape: (), Type: <dtype: 'float32'>
Tensor: 42, Shape: (), Type: <dtype: 'int32'>
Tensor: True, Shape: (), Type: <dtype: 'bool'>
Tensor: b'Hello, TensorFlow!', Shape: (), Type: <dtype: 'string'>
```

Tensor Operations

Common Mathematical Operations on tensors:

Operation	Example
Addition	`tf.add(tensor1, tensor2) or tensor1 + tensor2`
Subtraction	`tf.subtract(tensor1, tensor2) or tensor1 - tensor2`
Multiplication	`tf.multiply(tensor1, tensor2) or tensor1 * tensor2`
Matrix Multiplication	`tf.matmul(tensor1, tensor2)`
Dot Product	`tf.tensordot(tensor1, tensor2, axes=1)`
Reshaping	`tf.reshape(tensor, new_shape)`

Code Snippet

```python
import tensorflow as tf

# Define tensors
a = tf.constant([[1, 2], [3, 4]], dtype=tf.float32)
b = tf.constant([[5, 6], [7, 8]], dtype=tf.float32)

# Basic tensor operations
add = tf.add(a, b)  # or a + b
sub = tf.subtract(a, b)  # or a - b
mul = tf.multiply(a, b)  # or a * b (element-wise)
matmul = tf.matmul(a, b)  # or a @ b (matrix multiplication)
dot_product = tf.tensordot(a, b, axes=1)
reshaped = tf.reshape(a, (4, 1))  # Reshape to (4,1)

# Print results
print(f"Addition:\n{add.numpy()}\n")
print(f"Subtraction:\n{sub.numpy()}\n")
print(f"Element-wise Multiplication:\n{mul.numpy()}\n")
print(f"Matrix Multiplication:\n{matmul.numpy()}\n")
print(f"Dot Product:\n{dot_product.numpy()}\n")
print(f"Reshaped Tensor:\n{reshaped.numpy()}\n")
```

```
Addition:
[[ 6.  8.]
 [10. 12.]]

Subtraction:
[[-4. -4.]
 [-4. -4.]]

Element-wise Multiplication:
[[ 5. 12.]
 [21. 32.]]

Matrix Multiplication:
[[19. 22.]
 [43. 50.]]

Dot Product:
[[19. 22.]
 [43. 50.]]

Reshaped Tensor:
[[1.]
 [2.]
 [3.]
 [4.]]
```

Indexing & Slicing

We can extract parts of a tensor just like NumPy array.

Operation	Example
Access Element	`tensor[1, 2]`
Slice rows	`tensor[0:2, :]`
Slice columns	`tensor[:, 1:3]`
Extract single row	`tensor[1, :]`
Extract single column	`tensor[:, 0]`

Code Snippet

0s

```python
import tensorflow as tf

tensor = tf.constant([[10, 20, 30],
                      [40, 50, 60],
                      [70, 80, 90]])

# Extracting elements
first_row = tensor[0, :]
second_column = tensor[:, 1]

print(f"First Row: {first_row.numpy()}")
print(f"Second Column: {second_column.numpy()}")
```

```
First Row: [10 20 30]
Second Column: [20 50 80]
```

GPU/CPU Execution

TensorFlow automatically uses GPU if available otherwise, it defaults to CPU. However, we can manually check and set devices to ensure TensorFlow utilises the desired hardware.

We can manually check and set devices.

[2]

```python
import tensorflow as tf

print("TensorFlow version:", tf.__version__)
print("GPU Available:", tf.config.list_physical_devices('GPU'))
```

```
TensorFlow version: 2.18.0
GPU Available: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Set Memory Growth to Prevent TensorFlow from Consuming All GPU Memory

By default, TensorFlow tends to allocate all available GPU memory for itself, even if it doesn't need it immediately. This can be problematic if we've other processes running on the GPU or if we want to run multiple TensorFlow programs concurrently.

To prevent this, we can enable memory growth:

[3]

```python
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        print("Memory growth enabled for GPU.")
    except RuntimeError as e:
        print(e)  # Memory growth must be set before initializing GPUs
```

```
Memory growth enabled for GPU.
```

This ensures TensorFlow only uses the necessary amount of memory instead of reserving all GPU resources.