Part 1: Theoretical Quetions

1) What are the different data structures used in Tensorflow?. Give some examples.

2) How does the TensorFlow constant differ from a TensorFlow variable? Explain with an example?

3)Describe the process of matrix addition, multiplication, and elementDwise operations in TensorFlow.

Part 1: Theoretical Questions

Tc: Different Data Structures Used in TensorFlow

TensorFlow uses various data structures to represent and manipulate data. Some of the primary data structures include:

1. **Tensors**:
    - Multi-dimensional arrays with a uniform type.
    - Similar to NumPy arrays but with additional capabilities for GPU acceleration.
    - Example: `tf.constant([1, 2, 3])`
2. **Variables**:
    - Mutable data structures that can be updated during the execution of a TensorFlow program.
    - Used to represent weights in neural networks.
    - Example: `tf.Variable([1.0, 2.0, 3.0])`
3. **Datasets**:
    - Used for input pipelines to feed data into models.
    - Can be created from various sources like arrays, files, or other datasets.
    - Example: `tf.data.Dataset.from_tensor_slices([1, 2, 3])`
4. **SparseTensors**:
    - Efficient representation of tensors with a majority of zero elements.
    - Example: `tf.sparse.SparseTensor(indices=[[0, 0], [1, 2]], values=[1, 2], dense_shape=[3, 4])`

Cc: TensorFlow Constant vs. TensorFlow Variable

- **TensorFlow Constant**:

- Immutable data structures.
- Values are fixed and cannot be changed once created.
- Useful for storing fixed values, like configurations or predefined data.

Example:

```
import tensorflow as tf
const = tf.constant([1.0, 2.0, 3.0])
print(const)
```

- **TensorFlow Variable**:
  - Mutable data structures.
  - Values can be updated during the execution of the program.
  - Commonly used for model parameters like weights and biases which need to be updated during training.

Example:

```
var = tf.Variable([1.0, 2.0, 3.0])
var.assign([4.0, 5.0, 6.0])
print(var)
```

=c: Matrix Operations in TensorFlow

TensorFlow provides various functions for matrix operations, including addition, multiplication, and element-wise operations.

1. **Matrix Addition**:
   - Element-wise addition of two matrices.

   Example:

```
a = tf.constant([[1, 2], [3, 4]])
b = tf.constant([[5, 6], [7, 8]])
result = tf.add(a, b)
print(result)
```

2. **Matrix Multiplication**:
   - Standard matrix multiplication (dot product).

   Example:

```
a = tf.constant([[1, 2], [3, 4]])
b = tf.constant([[5, 6], [7, 8]])
result = tf.matmul(a, b)
print(result)
```

3. **Element-wise Operations**:
    - Operations applied element by element, such as multiplication or division.

   Example:

```
a = tf.constant([[1, 2], [3, 4]])
b = tf.constant([[5, 6], [7, 8]])
result = tf.multiply(a, b)  # Element-wise multiplication
print(result)
```

Summary

- **Data Structures**: TensorFlow uses tensors, variables, datasets, and sparse tensors.
- **Constants vs. Variables**: Constants are immutable, while variables are mutable and can be updated.
- **Matrix Operations**: TensorFlow supports matrix addition, multiplication, and element-wise operations with straightforward syntax.

Part 2 practical implementation

*Creating and Manipulating Matrices *

1) Create a normal matrix A with dimensions 2x2, using TensorFlow's random_normal function. Display the values of matrix .

```
import tensorflow as tf

# Create matrix A with dimensions 2x2 using random_normal
A = tf.random.normal([2, 2])
print("Matrix A:")
print(A)
```

```
Matrix A:
tf.Tensor(
[[-0.05802727 -1.05032   ]
 [-0.9455281  -0.4682873 ]], shape=(2, 2), dtype=float32)
```

2)Create a Gaussian matrix B with dimensions x, using TensorFlow's truncated_normal function. Display the values of matrix B

```
# Create matrix B with dimensions 2x2 using truncated_normal
B = tf.random.truncated_normal([2, 2])
print("Matrix B:")
print(B)
```

```
Matrix B:
tf.Tensor(
[[-0.8247807   0.5296175 ]
 [-0.35310888 -0.16082053]], shape=(2, 2), dtype=float32)
```

3) Create a matrix C with dimensions 2x2, where the values are drawn from a normal distribution with a mean of 3 and a standard deviation of 0.5, using TensorFlow's random.normal function. Display the values of matrix.

```
# Create matrix C with dimensions 2x2 using random.normal with mean=2 and stddev=0.5
C = tf.random.normal([2, 2], mean=3, stddev=0.5)
print("Matrix C:")
print(C)
```

```
Matrix C:
tf.Tensor(
[[2.202033  3.2501633]
 [3.2113674 3.6343849]], shape=(2, 2), dtype=float32)
```

```
##4)Perform matrix addition between matrix A and matrix B, and store the result in matrix D.
# Perform matrix addition between A and B
D = tf.add(A, B)
print("Matrix D (A + B):")
print(D)
```

```
Matrix D (A + B):
tf.Tensor(
[[-0.88280797 -0.52070254]
 [-1.2986369  -0.62910783]], shape=(2, 2), dtype=float32)
```

```
##5Perform matrix multiplication between matrix C and matrix D, and store the result in matr
# Perform matrix multiplication between C and D
E = tf.matmul(C, D)
print("Matrix E (C * D):")
print(E)

#
```

```
Matrix E (C * D):
tf.Tensor(
[[-6.164755  -3.1913075]
 [-7.5547667 -3.9585872]], shape=(2, 2), dtype=float32)
```

Task 2: Performing Additional Matrix Operations

```
import tensorflow as tf

# Create matrix F with dimensions 2x2 using random_uniform
F = tf.random.uniform([2, 2])
print("Matrix F:")
print(F)

# Calculate the transpose of matrix F
G = tf.transpose(F)
print("Matrix G (transpose of F):")
print(G)

# Calculate the element-wise exponential of matrix F
H = tf.exp(F)
print("Matrix H (element-wise exponential of F):")
print(H)

# Concatenate matrix F and matrix G horizontally
I = tf.concat([F, G], axis=1)
print("Matrix I (F and G concatenated horizontally):")
print(I)

# Concatenate matrix F and matrix H vertically
J = tf.concat([F, H], axis=0)
print("Matrix J (F and H concatenated vertically):")
print(J)


Start coding or generate with AI.
```

```
[[0.7593688 0.4971155]
 [0.9994184 0.8771013]], shape=(2, 2), dtype=float32)
Matrix G (transpose of F):
tf.Tensor(
[[0.7593688 0.9994184]
 [0.4971155 0.8771013]], shape=(2, 2), dtype=float32)
Matrix H (element-wise exponential of F):
tf.Tensor(
[[2.136927  1.6439724]
 [2.7167013 2.4039214]], shape=(2, 2), dtype=float32)
```