100 Days with TensorFlow - One Day Per Day!

This document summarizes the 100 lessons on learning TensorFlow, one lesson each day. The difficulty will gradually increase, with weekly progress reviews. The goal is to build a solid foundation in TensorFlow and work on a final project together. Each lesson includes code examples and explanations.

Lesson 1: Introduction to TensorFlow Basics

In this lesson, we introduced TensorFlow and its use of tensors for handling data. Tensors are the fundamental building blocks in TensorFlow. A tensor is a generalization of vectors and matrices to potentially higher dimensions. TensorFlow represents all data in this multi-dimensional array format.

We started by printing the TensorFlow version and creating two constant tensors, using `tf.constant()`. Tensors in TensorFlow are immutable (their values can't change), and they can store data of any type like integers, floats, strings, or booleans.

We then performed a basic addition operation using `tf.add()`. TensorFlow provides built-in functions for performing mathematical operations on tensors, such as addition, multiplication, and matrix operations. The result of an operation is returned as another tensor.

In Part 2, we explored tensor shapes. A tensor's shape defines its dimensionality, or how many elements it contains in each dimension. A scalar (0-D tensor) has no dimensions, while a vector (1-D tensor) has one. A matrix (2-D tensor) has two dimensions, and higher-dimensional tensors can represent more complex data structures like batches of images.

Reshaping tensors is an important concept in TensorFlow. It allows you to change the shape of the

tensor without changing its underlying data. For example, a tensor with 4 elements can be reshaped into a 2x2 matrix or a 1D vector. However, the total number of elements must remain the same.

In Part 3, we applied various tensor operations. TensorFlow supports arithmetic operations like addition, subtraction, multiplication, and division directly on tensors. We also performed matrix multiplication using `tf.matmul()`, which is useful in neural networks, where weights and inputs are often represented as matrices.

Finally, we discussed why reshaping is important. Reshaping tensors helps adjust the data's structure for specific machine learning models, batch processing, or to meet the input/output requirements of layers in neural networks.

Fun Facts

Did you know? When reshaping tensors, the total number of elements must remain the same! For example, a tensor with 4 elements can be reshaped into (2, 1, 2), but not (2, 1, 1), since the number of elements would not match. Tensors can be reshaped to fit various dimensions based on how you need to structure the data.

Code Used in Lesson 1

```
# Lesson 1, Part 1 - Basic TensorFlow Program
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
tensor_a = tf.constant(5)
tensor_b = tf.constant(3)
result = tf.add(tensor_a, tensor_b)
```

```
print("Result of addition:", result)


# Lesson 1, Part 2 - Tensor Types and Shapes

int_tensor = tf.constant(10, dtype=tf.int32)

float_tensor = tf.constant(10.5, dtype=tf.float32)

string_tensor = tf.constant("Hello TensorFlow")

bool_tensor = tf.constant(True)

print("Integer Tensor:", int_tensor)

print("Float Tensor:", float_tensor)

print("String Tensor:", string_tensor)

print("Boolean Tensor:", bool_tensor)


# Reshaping a tensor

reshaped_tensor = tf.reshape(tensor_b, (2, 1))

print("Reshaped Tensor:", reshaped_tensor)


# Lesson 1, Part 3 - Tensor Operations

add_result = tf.add(tensor_a, tensor_b)

multiply_result = tf.multiply(tensor_a, tensor_b)

subtract_result = tf.subtract(tensor_a, tensor_b)

divide_result = tf.divide(tensor_a, tensor_b)

print("Addition Result:", add_result)

print("Multiplication Result:", multiply_result)

print("Subtraction Result:", subtract_result)

print("Division Result:", divide_result)
```

```
# Matrix multiplication

matrix_1 = tf.constant([[1, 2], [3, 4]])

matrix_2 = tf.constant([[5, 6], [7, 8]])

matrix_mult_result = tf.matmul(matrix_1, matrix_2)

print("Matrix Multiplication Result:", matrix_mult_result)


# Other operations

sum_result = tf.reduce_sum(matrix_1)

mean_result = tf.reduce_mean(matrix_1)

max_index = tf.argmax(tensor_b)

print("Sum of elements:", sum_result)

print("Mean of elements:", mean_result)

print("Index of max value:", max_index)
```

Lesson 2: TensorFlow Operations and Variables

In this lesson, we explored TensorFlow variables, operations, and broadcasting. We started by learning how to create variables in TensorFlow using `tf.Variable()`, which allows for mutable values, unlike constant tensors. Variables can be updated with methods like `.assign()`.

Next, we delved into basic operations beyond addition, such as matrix multiplication (`tf.matmul()`) and element-wise multiplication (`tf.multiply()`), to perform matrix operations commonly needed in machine learning and neural networks.

Finally, we learned about broadcasting, a powerful TensorFlow feature that allows tensors of different shapes to be used together in element-wise operations. Broadcasting stretches the smaller tensor to match the shape of the larger tensor, allowing for efficient operations.

Fun Facts

Matrix multiplication and element-wise multiplication serve different purposes. Matrix multiplication combines rows and columns and is used in applications such as neural networks and transformations. Element-wise multiplication operates directly on individual elements, useful for scaling or masking. Both are key operations in machine learning.

Matrix Multiplication vs. Element-wise Multiplication

Matrix multiplication (dot product) involves combining rows of one matrix with columns of another. It is widely used in neural networks.

Element-wise multiplication multiplies corresponding elements of two matrices directly. Both require specific rules:

Matrix Multiplication:

- Requires the number of columns in the first matrix to equal the number of rows in the second.

- Produces a new matrix whose shape is determined by the outer dimensions of the input matrices.

Element-wise Multiplication:

- Requires matrices to have the same shape.

- Multiplies corresponding elements of the two matrices directly.

Code Examples for Practice

```
# 1. Matrix Multiplication
import tensorflow as tf
matrix_a = tf.constant([[1, 2], [3, 4]])
matrix_b = tf.constant([[5, 6], [7, 8]])
result = tf.matmul(matrix_a, matrix_b)
print(result.numpy())

# 2. Element-wise Multiplication
elementwise_product = tf.multiply(matrix_a, matrix_b)
print(elementwise_product.numpy())

# 3. Broadcasting Example
tensor_a = tf.constant([1, 2])
tensor_b = tf.constant([[1, 2], [3, 4]])
broadcast_result = tf.add(tensor_a, tensor_b)
```

```
print(broadcast_result.numpy())
```