**LECTURE**

**https://github.com/JNYH/DataCamp\_Introduction\_to\_Tensorflow\_in\_Python/blob/4daba9dd303c2ca3317ea906e8b4f059dd214ecd/Course\_notes\_solutions\_answers\_Introduction\_to\_Tensorflow\_in\_Python.pdf**

**1. Constants and variables**

00:00 - 00:18

Hi! My name is Isaiah Hull and this is a course on the fundamentals of the TensorFlow API in Python. In our first video, we will briefly introduce TensorFlow and then discuss its two basic objects of computation: constants and variables.

**2. What is TensorFlow?**

00:18 - 00:56

TensorFlow is an open-source library for graph-based numerical computation. It was developed by the Google Brain Team. It has both low and high level APIs. You can use TensorFlow to perform addition, multiplication, and differentiation. You can also use it to design and train machine learning models. TensorFlow two point zero brought with it substantial changes. Eager execution is now enabled by default, which allows users to write simpler and more intuitive code. Additionally, model building is now centered around the Keras and Estimators high-level APIs.

**3. What is a tensor?**

00:56 - 01:14

The TensorFlow documentation describes a tensor as "a generalization of vectors and matrices to potentially higher dimensions." Now, if you are not familiar with linear algebra, you can simply think of a tensor as a collection of numbers, which is arranged into a particular shape.

**4. What is a tensor?**

01:14 - 01:39

As an example, let's say you have a slice of bread and you cut it into 9 pieces. One of those 9 pieces is a 0-dimensional tensor. This corresponds to a single number. A collection of 3 pieces that form a row or column is a 1-dimensional tensor. All 9 pieces together are a 2-dimensional tensor. And the whole loaf, which contains many slices, is a 3-dimensional tensor.

**5. Defining tensors in TensorFlow**

01:39 - 01:56

Now that you know what a tensor is, let's define a few. We will start by importing tensorflow as tf. We will then define 0-, 1-, 2-, and 3-dimensional tensors. Note that each object will be a tf dot Tensor object.

**6. Defining tensors in TensorFlow**

01:56 - 02:06

If we want to print the array contained in that object, we can apply the dot numpy method and pass the resulting object to the print function.

**7. Defining constants in TensorFlow**

02:06 - 02:28

We next move on to constants, which are the simplest category of tensor in TensorFlow. A constant does not change and cannot be trained. It can, however, have any dimension. In the code block, we've defined two constants. The constant a is a 2x3 tensor of 3s. The constant b is a 2x2 tensor, which is constructed from the 1-dimensional tensor: 1, 2, 3, 4.

**8. Using convenience functions to define constants**

02:28 - 03:06

In the previous slide, we worked exclusively with the constant operation. However, in some cases, there are more convenient options for defining certain types of special tensors. You can use the zeros or ones operations to generate a tensor of arbitrary dimension that is populated entirely with zeros or ones. You can use the zeros\_like or ones\_like operations to populate tensors with zeros and ones, copying the dimension of some input tensor. Finally, you can use the fill operation to populate a tensor of arbitrary dimension with the same scalar value in each element.

**9. Defining and initializing variables**

03:06 - 03:52

Unlike a constant, a variable's value can change during computation. The value of a variable is shared, persistent, and modifiable. However, its data type and shape are fixed. Let's take a look at how variables are constructed and used in TensorFlow. In the code, we first define a variable, a0, which is a 1-dimensional tensor with 6 elements. We can set its datatype to a 32-bit float or something else, such as a 16-bit int, as we have for a1. We then define a constant, b. And define c0 as the product of a0 and b. Note that certain TensorFlow operations, such as tf.multiply are overloaded, which allows us to use the simpler a0\*b expression instead.

**10. Let's practice!**

03:52 - 03:57

It's now time to put what you've learned to use in some exercises.

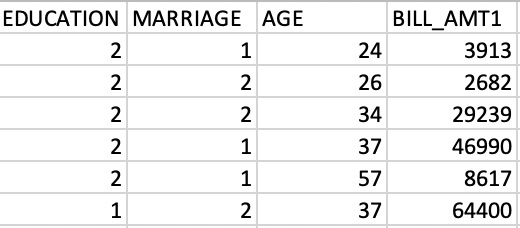
## Exercise

# Defining data as constants

Throughout this course, we will use tensorflow version 2.6.0 and will exclusively import the submodules needed to complete each exercise. This will usually be done for you, but you will do it in this exercise by importing constant from tensorflow.

After you have imported constant, you will use it to transform a numpy array, credit\_numpy, into a tensorflow constant, credit\_constant. This array contains feature columns from a dataset on credit card holders and is previewed in the image below. We will return to this dataset in later chapters.

Note that tensorflow 2 allows you to use data as either a numpy array or a tensorflow constant object. Using a constant will ensure that any operations performed with that object are done in tensorflow.



## Instructions

100 XP

* Import the constant submodule from the tensorflow module.
* Convert the credit\_numpy array into a constant object in tensorflow. Do not set the data type.

# Import constant from TensorFlow

from tensorflow import \_\_\_\_

# Convert the credit\_numpy array into a tensorflow constant

credit\_constant = constant(\_\_\_\_)

# Print constant datatype

print('\n The datatype is:', credit\_constant.dtype)

# Print constant shape

print('\n The shape is:', credit\_constant.shape)

# Import constant from TensorFlow

from tensorflow import constant

# Convert the credit\_numpy array into a tensorflow constant

credit\_constant = constant(credit\_numpy)

# Print constant datatype

print('\n The datatype is:', credit\_constant.dtype)

# Print constant shape

print('\n The shape is:', credit\_constant.shape)

**The datatype is: <dtype: 'float64'>**

**The shape is: (30000, 4)**

# Import constant from TensorFlow

from tensorflow import constant

# Convert the credit\_numpy array into a tensorflow constant

credit\_constant = constant(credit\_numpy)

# Print constant datatype

print('\n The datatype is:', credit\_constant.dtype)

# Print constant shape

print('\n The shape is:', credit\_constant.shape)

Excellent! You now understand how constants are used in tensorflow. In the following exercise, you'll practice defining variables.

## Exercise

# Defining variables

Unlike a constant, a variable's value can be modified. This will be useful when we want to train a model by updating its parameters.

Let's try defining and printing a variable. We'll then convert the variable to a numpy array, print again, and check for differences. Note that Variable(), which is used to create a variable tensor, has been imported from tensorflow and is available to use in the exercise.

## Instructions

* Define a variable, A1, as the 1-dimensional tensor: [1, 2, 3, 4].

# Convert A1 to a numpy array and assign it to B1

B1 = A1.numpy()

# Print B1

print('\n B1: ', B1)

Apply .numpy() to A1 and assign it to B1.

# Define the 1-dimensional variable A1

A1 = Variable([1, 2, 3, 4])

# Print the variable A1

print('\n A1: ', A1)

# Define the 1-dimensional variable A1

A1 = Variable([1, 2, 3, 4])

# Print the variable A1

print('\n A1: ', A1)

# Convert A1 to a numpy array and assign it to B1

B1 = A1.numpy()

# Print B1

print('\n B1: ', B1)

A1: <tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4], dtype=int32)>

B1: [1 2 3 4]

Nice work! Did you notice any differences between the print statements for A1 and B1? In our next exercise, we'll review how to check the properties of a tensor after it is already defined.

**1. Basic operations**

00:00 - 00:05

In this video, we'll talk about basic operations in TensorFlow.

**2. What is a TensorFlow operation?**

00:05 - 00:17

TensorFlow has a model of computation that revolves around the use of graphs. A TensorFlow graph contains edges and nodes, where the edges are tensors and the nodes are operations.

**3. What is a TensorFlow operation?**

00:17 - 00:28

In the graph shown, which was drawn using TensorFlow, the const operations define 2 by 2 constant tensors. Two tensors are summed using the add operation.

**4. What is a TensorFlow operation?**

00:28 - 00:33

Another two tensors are then summed using the add operation.

**5. What is a TensorFlow operation?**

00:33 - 00:38

Finally, the resulting matrices are multiplied together with the matmul operation.

**6. Applying the addition operator**

00:38 - 00:53

Let's start with the addition operator. We will first import the constant and add operations. We may now use constant to define 0-dimensional, 1-dimensional, and 2-dimensional tensors.

**7. Applying the addition operator**

00:53 - 01:03

Finally, let's add them together using the operation for tensor addition. Note that we can perform scalar addition with A0 and B0, vector addition with A1 and B1, and matrix addition with A2 and B2.

**8. Performing tensor addition**

01:03 - 01:42

The add operation performs element-wise addition with two tensors. Each pair of tensors added must have the same shape. Element-wise addition of the scalars 1 and 2 yields the scalar 3. Element-wise addition of the vectors 1,2 and 3,4 yields the vector 4,6. Element-wise addition of the matrices 1,2,3,4 and 5,6,7,8 yields the matrix 6,8,10,12. Furthermore, the add operator is overloaded, which means that we can also perform addition using the plus symbol.

**9. How to perform multiplication in TensorFlow**

01:42 - 02:16

We will consider both element-wise and matrix multiplication. For element-wise multiplication, which is performed with the multiply operation, the tensors involved must have the same shape. For instance, you may want to multiply the vector 1,2,3 by 3,4,5 or 1,2 by 3,4. For matrix multiplication, you use the matmul operator. Note that performing matmul(A,B) requires that the number of columns of A equal the number of rows of B.

**10. Applying the multiplication operators**

02:16 - 03:01

Let's look at some examples of multiplication in TensorFlow. We'll import the ones operator, along with the two types of multiplication we will use. We will also define a scalar, A0, a 3 by 1 vector of ones, a 3 by 4 vector of ones, and a 4 by 3 vector of ones. What operations can be performed using these tensors of ones? We can perform element-wise multiplication of any element by itself, such as A0 by A0, A31 by A31, or A34 by A34. We can also perform matrix multiplication of A43 by A34, but not A43 by A43.

**11. Summing over tensor dimensions**

03:01 - 03:25

Finally, we end this lesson by discussing summation over tensors, which is performed using the reduce sum operator. This can be used to sum over all dimensions of a tensor or just one. Let's see how this works in practice. We will import ones and reduce sum from tensorflow. We will then define a 2 by 3 by 4 tensor that consists of ones.

**12. Summing over tensor dimensions**

03:25 - 03:52

If we sum over all elements of A, we get 24, since the tensor contains 24 elements, all of which are 1. If we sum over dimension 0, we get a 3 by 4 matrix of 2s. If we sum over 1, we get a 2 by 4 matrix of 3s. And if we sum over 2, we get a 2 by 3 matrix of 4s. In each case, we reduce the size of the tensor by summing over one of its dimensions.

**13. Let's practice!**

03:52 - 04:01

Now that you understand how to perform basic operations in TensorFlow, let's put this to work with some exercises.

## Exercise

# Performing element-wise multiplication

Element-wise multiplication in TensorFlow is performed using two tensors with identical shapes. This is because the operation multiplies elements in corresponding positions in the two tensors. An example of an element-wise multiplication, denoted by the

symbol, is shown below:

In this exercise, you will perform element-wise multiplication, paying careful attention to the shape of the tensors you multiply. Note that multiply(), constant(), and ones\_like() have been imported for you.

## Instructions

* Define the tensors A1 and A23 as constants.
* Set B1 to be a tensor of ones with the same shape as A1.
* Set B23 to be a tensor of ones with the same shape as A23.
* Set C1 and C23 equal to the element-wise products of A1 and B1, and A23 and B23, respectively.

**# Define tensors A1 and A23 as constants**

**A1 = constant([1, 2, 3, 4])**

**A23 = constant([[1, 2, 3], [1, 6, 4]])**

**# Define B1 and B23 to have the correct shape**

**B1 = ones\_like(A1)**

**B23 = ones\_like(A23)**

**# Perform element-wise multiplication**

**C1 = multiply(A1, B1)**

**C23 = multiply(A23, B23)**

**# Print the tensors C1 and C23**

**print('\n C1: {}'.format(C1.numpy()))**

**print('\n C23: {}'.format(C23.numpy()))**

**# Define tensors A1 and A23 as constants**

**A1 = constant([1, 2, 3, 4])**

**A23 = constant([[1, 2, 3], [1, 6, 4]])**

**# Define B1 and B23 to have the correct shape**

**B1 = ones\_like(A1)**

**B23 = ones\_like(A23)**

**# Perform element-wise multiplication**

**C1 = multiply(A1, B1)**

**C23 = multiply(A23, B23)**

**# Print the tensors C1 and C23**

**print('\n C1: {}'.format(C1.numpy()))**

**print('\n C23: {}'.format(C23.numpy()))**

**C1: [1 2 3 4]**

**C23: [[1 2 3]**

**[1 6 4]]**

**Excellent work! Notice how performing element-wise multiplication with tensors of ones leaves the original tensors unchanged.**

## Exercise

# Making predictions with matrix multiplication

In later chapters, you will learn to train linear regression models. This process will yield a vector of parameters that can be multiplied by the input data to generate predictions. In this exercise, you will use input data, features, and a target vector, bill, which are taken from a credit card dataset we will use later in the course.

, ,

The matrix of input data, features, contains two columns: education level and age. The target vector, bill, is the size of the credit card borrower's bill.

Since we have not trained the model, you will enter a guess for the values of the parameter vector, params. You will then use matmul() to perform matrix multiplication of features by params to generate predictions, billpred, which you will compare with bill. Note that we have imported matmul() and constant().

## Instructions

* Define features, params, and bill as constants.
* Compute the predicted value vector, billpred, by multiplying the input data, features, by the parameters, params. Use matrix multiplication, rather than the element-wise product.
* Define error as the targets, bill, minus the predicted values, billpred.

# Define features, params, and bill as constants

features = \_\_\_\_([[2, 24], [2, 26], [2, 57], [1, 37]])

params = \_\_\_\_([[1000], [150]])

bill = \_\_\_\_([[3913], [2682], [8617], [64400]])

# Compute billpred using features and params

billpred = \_\_\_\_

# Compute and print the error

error = \_\_\_\_ - \_\_\_\_

print(error.numpy())

# Define features, params, and bill as constants

features = constant([[2, 24], [2, 26], [2, 57], [1, 37]])

params = constant([[1000], [150]])

bill = constant([[3913], [2682], [8617], [64400]])

# Compute billpred using features and params

billpred = matmul(features, params)

# Compute and print the error

error = bill - billpred

print(error.numpy())

# Define features, params, and bill as constants

features = constant([[2, 24], [2, 26], [2, 57], [1, 37]])

params = constant([[1000], [150]])

bill = constant([[3913], [2682], [8617], [64400]])

# Compute billpred using features and params

billpred = matmul(features, params)

# Compute and print the error

error = bill - billpred

print(error.numpy())

[[-1687]

[-3218]

[-1933]

[57850]]

**Nice job! Understanding matrix multiplication will make things simpler when we start making predictions with linear models.**

## Exercise

# Summing over tensor dimensions

You've been given a matrix, wealth. This contains the value of bond and stock wealth for five individuals in thousands of dollars.

wealth =

The first column corresponds to bonds and the second corresponds to stocks. Each row gives the bond and stock wealth for a single individual. Use wealth, reduce\_sum(), and .numpy() to determine which statements are correct about wealth.

## Instructions

### Possible answers

The individual in the first row has the highest total weath (ie stocks +bonds)

Combined, the 5 individuals hold %50000 in stocks.

Combined, the 5 individuals hold $50000 in bonds.

The individual in the second row has the lowest total wealth (ie stocks + bon



