

C2W1_Assignment

December 20, 2020

1 Basic Tensor operations and GradientTape.

In this graded assignment, you will perform different tensor operations as well as use [GradientTape](#). These are important building blocks for the next parts of this course so it's important to master the basics. Let's begin!

```
[1]: import tensorflow as tf
import numpy as np
```

1.1 Exercise 1 - `tf.constant`

Creates a constant tensor from a tensor-like object.

```
[2]: # Convert NumPy array to Tensor using `tf.constant`
def tf_constant(array):
    """
    Args:
        array (numpy.ndarray): tensor-like array.

    Returns:
        tensorflow.python.framework.ops.EagerTensor: tensor.
    """
    ### START CODE HERE ###
    tf_constant_array = tf.constant(array)
    ### END CODE HERE ###
    return tf_constant_array
```

```
[3]: tmp_array = np.arange(1,10)
x = tf_constant(tmp_array)
x

# Expected output:
# <tf.Tensor: shape=(9,), dtype=int64, numpy=array([1, 2, 3, 4, 5, 6, 7, 8, 9])>
```

```
[3]: <tf.Tensor: shape=(9,), dtype=int64, numpy=array([1, 2, 3, 4, 5, 6, 7, 8, 9])>
```

Note that for future docstrings, the type `EagerTensor` will be used as a shortened version of `tensorflow.python.framework.ops.EagerTensor`.

1.2 Exercise 2 - `tf.square`

Computes the square of a tensor element-wise.

```
[4]: # Square the input tensor
def tf_square(array):
    """
    Args:
        array (numpy.ndarray): tensor-like array.

    Returns:
        EagerTensor: tensor.
    """
    # make sure it's a tensor
    array = tf.constant(array)

    ### START CODE HERE ###
    tf_squared_array = tf.square(array)
    ### END CODE HERE ###
    return tf_squared_array
```

```
[5]: tmp_array = tf.constant(np.arange(1, 10))
x = tf_square(tmp_array)
x

# Expected output:
# <tf.Tensor: shape=(9,), dtype=int64, numpy=array([ 1,  4,  9, 16, 25, 36, 49, 64, 81])>
```

```
[5]: <tf.Tensor: shape=(9,), dtype=int64, numpy=array([ 1,  4,  9, 16, 25, 36, 49, 64, 81])>
```

1.3 Exercise 3 - `tf.reshape`

Reshapes a tensor.

```
[6]: # Reshape tensor into a 3 x 3 matrix
def tf_reshape(array, shape):
    """
    Args:
        array (EagerTensor): tensor to reshape.
        shape (tuple): desired shape.
```

```

Returns:
    EagerTensor: reshaped tensor.
"""
# make sure it's a tensor
array = tf.constant(array)
### START CODE HERE ###
tf_resaped_array = tf.reshape(array, shape)
### END CODE HERE ###
return tf_resaped_array

```

```

[7]: # Check your function
tmp_array = np.array([1,2,3,4,5,6,7,8,9])
# Check that your function reshapes a vector into a matrix
x = tf_reshape(tmp_array, (3, 3))
x

# Expected output:
# <tf.Tensor: shape=(3, 3), dtype=int64, numpy=
# [[1, 2, 3],
#  [4, 5, 6],
#  [7, 8, 9]]

```

```

[7]: <tf.Tensor: shape=(3, 3), dtype=int64, numpy=
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])>

```

1.4 Exercise 4 - `tf.cast`

Casts a tensor to a new type.

```

[8]: # Cast tensor into float32
def tf_cast(array, dtype):
    """
    Args:
        array (EagerTensor): tensor to be casted.
        dtype (tensorflow.python.framework.dtypes.DType): desired new type.
        ↪ (Should be a TF dtype!)

    Returns:
        EagerTensor: casted tensor.
    """
    # make sure it's a tensor
    array = tf.constant(array)

    ### START CODE HERE ###

```

```

tf_cast_array = tf.cast(array, dtype)
### END CODE HERE ###
return tf_cast_array

```

```

[9]: # Check your function
tmp_array = [1,2,3,4]
x = tf_cast(tmp_array, tf.float32)
x

# Expected output:
# <tf.Tensor: shape=(4,), dtype=float32, numpy=array([1., 2., 3., 4.],
↳dtype=float32)>

```

```

[9]: <tf.Tensor: shape=(4,), dtype=float32, numpy=array([1., 2., 3., 4.],
dtype=float32)>

```

1.5 Exercise 5 - `tf.multiply`

Returns an element-wise $x * y$.

```

[10]: # Multiply tensor1 and tensor2
def tf_multiply(tensor1, tensor2):
    """
    Args:
        tensor1 (EagerTensor): a tensor.
        tensor2 (EagerTensor): another tensor.

    Returns:
        EagerTensor: resulting tensor.
    """
    # make sure these are tensors
    tensor1 = tf.constant(tensor1)
    tensor2 = tf.constant(tensor2)

    ### START CODE HERE ###
    product = tf.multiply(tensor1, tensor2)
    ### END CODE HERE ###
    return product

```

```

[11]: # Check your function
tmp_1 = tf.constant(np.array([[1,2],[3,4]]))
tmp_2 = tf.constant(np.array(2))
result = tf_multiply(tmp_1, tmp_2)
result

# Expected output:

```

```
# <tf.Tensor: shape=(2, 2), dtype=int64, numpy=
# array([[2, 4],
#        [6, 8]])>
```

```
[11]: <tf.Tensor: shape=(2, 2), dtype=int64, numpy=
      array([[2, 4],
             [6, 8]])>
```

1.6 Exercise 6 - `tf.add`

Returns $x + y$ element-wise.

```
[12]: # Add tensor1 and tensor2
def tf_add(tensor1, tensor2):
    """
    Args:
        tensor1 (EagerTensor): a tensor.
        tensor2 (EagerTensor): another tensor.

    Returns:
        EagerTensor: resulting tensor.
    """
    # make sure these are tensors
    tensor1 = tf.constant(tensor1)
    tensor2 = tf.constant(tensor2)

    ### START CODE HERE ###
    total = tf.add(tensor1, tensor2)
    ### END CODE HERE ###
    return total
```

```
[13]: # Check your function
tmp_1 = tf.constant(np.array([1, 2, 3]))
tmp_2 = tf.constant(np.array([4, 5, 6]))
tf_add(tmp_1, tmp_2)

# Expected output:
# <tf.Tensor: shape=(3,), dtype=int64, numpy=array([5, 7, 9])>
```

```
[13]: <tf.Tensor: shape=(3,), dtype=int64, numpy=array([5, 7, 9])>
```

1.7 Exercise 7 - Gradient Tape

Implement the function `tf_gradient_tape` by replacing the instances of `None` in the code below. The instructions are given in the code comments.

You can review the [docs](#) or revisit the lectures to complete this task.

```
[14]: def tf_gradient_tape(x):  
    """  
    Args:  
        x (EagerTensor): a tensor.  
  
    Returns:  
        EagerTensor: Derivative of z with respect to the input tensor x.  
    """  
    with tf.GradientTape() as t:  
  
        ### START CODE HERE ###  
        # Record the actions performed on tensor x with `watch`  
        t.watch(x)  
  
        # Define a polynomial of form  $3x^3 - 2x^2 + x$   
        y = 3*x**3 - 2*x**2 + x  
  
        # Obtain the sum of the elements in variable y  
        z = tf.reduce_sum(y)  
  
        # Get the derivative of z with respect to the original input tensor x  
        dz_dx = t.gradient(z, x)  
        ### END CODE HERE  
  
    return dz_dx
```

```
[15]: # Check your function  
tmp_x = tf.constant(2.0)  
dz_dx = tf_gradient_tape(tmp_x)  
result = dz_dx.numpy()  
result  
  
# Expected output:  
# 29.0
```

```
[15]: 29.0
```

Congratulations on finishing this week's assignment!

Keep it up!