# Assignment 1.1

## Tensor Operations in TensorFlow and PyTorch

In this assignment, you will focus on developing and implementing math operations on tensors of different dimensions in TensorFlow and PyTorch. You will be developing and implementing basic math operations on one-, two-, and three-dimensional tensors in TensorFlow and PyTorch. This will include creating and manipulating tensors, performing arithmetic operations on tensors, and exploring the use of functions that operate on tensors. For this assignment, you can use a Jupyter Notebook or a Python script.

---

### Part 1 of the Assignment: Tensor Operations

1. Create a one-dimensional tensor in both TensorFlow and PyTorch that contains the values of your interest. Print the tensor to the console for each framework.
2. Create a two-dimensional tensor in both TensorFlow and PyTorch that contains the values of your interest. Print the tensor to the console for each framework.
3. Create a three-dimensional tensor in both TensorFlow and PyTorch that contains the values of your interest. Print the tensor to the console for each framework.
4. Create a function in both TensorFlow and PyTorch that takes two tensors as input and returns the sum of their elements. Test the function with 1D, 2D, and 3D tensors. This function will demonstrate how to perform element-wise addition between two tensors and then sum all elements of the resulting tensor.

---

### Part 2 of the Assignment: The Role of Broadcasting in Tensor Operations

- Broadcasting is a powerful mechanism that allows TensorFlow and PyTorch to work with tensors of different shapes during arithmetic operations.

1. Describe how broadcasting works in tensor operations, providing a simple example to illustrate your explanation.
2. Discuss why broadcasting is important and how it enhances the flexibility of tensor operations in building neural network models.
3. Explain what limitations or challenges might arise when relying on broadcasting, particularly in the context of ensuring model correctness and efficiency.

In [ ]:
```
# importing tensorflow keras numpy and pytorch
import tensorflow as tf
from tensorflow import keras
```

```
import numpy as np
import torch
```

## Tensors in TF and PyTorch

In [ ]:
```
# creating a 1D tensor in tensorflow
tf_1d = tf.constant([1, 2, 3, 4, 5])

# creating a 2D tensor in tensorflow
tf_2d = tf.constant([[1, 2, 3], [4, 5, 6]])

# creating a 3d tensor in tensorflow
tf_3d = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

In [ ]:
```
# creating a 1D tensor in pytorch
d2_t = torch.tensor([1, 2, 3, 4, 5])

# createing a 2D tensor in pytorch
d2_t = torch.tensor([[1, 2, 3], [4, 5, 6]])

# creating a 3D tensor in pytorch
d3_t = torch.tensor([[[1, 2, 3], [4, 5, 6], [7, 8, 9]]])
```

## Tensorflow function

In [ ]:
```
# tensor flow function

def tf_function(tensor1, tensor2):
    summed_tensor = tf.add(tensor1, tensor2)
    return tf.reduce_sum(summed_tensor)
```

In [ ]:
```
# testing the function
tensor_1d_tf = tf.constant([1, 2, 3, 4, 5])
tensor_1d_tf_2 = tf.constant([5, 4, 3, 2, 1])

tensor_2d_tf = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
tensor_2d_tf_2 = tf.constant([[9, 8, 7], [6, 5, 4], [3, 2, 1]])

tensor_3d_tf = tf.constant([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
tensor_3d_tf_2 = tf.constant([[[12, 11, 10], [9, 8, 7]], [[6, 5, 4], [3, 2, 1]]])

# printing the results
print("the sum of the 1D tensor is: ", tf_function(tensor_1d_tf, tensor_1d_tf_2))
print("the sum of the 2D tensor is: ", tf_function(tensor_2d_tf, tensor_2d_tf_2))
print("the sum of the 3D tensor is: ", tf_function(tensor_3d_tf, tensor_3d_tf_2))
```

```
the sum of the 1D tensor is:  tf.Tensor(30, shape=(), dtype=int32)
the sum of the 2D tensor is:  tf.Tensor(90, shape=(), dtype=int32)
the sum of the 3D tensor is:  tf.Tensor(156, shape=(), dtype=int32)
```

## Pytorch function

In [ ]:
```python
# pytorch function

def torch_function(tensor1, tensor2):
    summed_tensor = torch.add(tensor1, tensor2)
    return torch.sum(summed_tensor)
```

In [ ]:
```python
# testing the function
tensor_1d_torch = torch.tensor([1, 2, 3, 4, 5])
tensor_1d_torch_2 = torch.tensor([5, 4, 3, 2, 1])

tensor_2d_torch = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
tensor_2d_torch_2 = torch.tensor([[9, 8, 7], [6, 5, 4], [3, 2, 1]])

tensor_3d_torch = torch.tensor([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
tensor_3d_torch_2 = torch.tensor([[[12, 11, 10], [9, 8, 7]], [[6, 5, 4], [3, 2, 1]]

# printing the results
print("the sum of the 1D tensor is: ", torch_function(tensor_1d_torch, tensor_1d_to
print("the sum of the 2D tensor is: ", torch_function(tensor_2d_torch, tensor_2d_to
print("the sum of the 3D tensor is: ", torch_function(tensor_3d_torch, tensor_3d_to
```

```
the sum of the 1D tensor is:  tensor(30)
the sum of the 2D tensor is:  tensor(90)
the sum of the 3D tensor is:  tensor(156)
```

# Broadcasting in Tensor Operations

---

`1. Describe how broadcasting works in tensor operations, providing a simple example to illustrate your explanation.`

- Broadcasting in tensor operations is when one tensor is expanded to match the shape of another tensor so that the operation can be performed. The tensors are then comared element-wise from the trailing dimension. If the dimensions are equal or one of the dimensions is 1, the tensors are compatible. If the dimensions are not equal and one of the dimensions is not 1, the tensors are not compatible.

`2. Discuss why broadcasting is important and how it enhances the flexibility of tensor operations in building neural network models.`

- The advantage of broadcasting is that it allows for the flexibility of working with tensors of different shapes during arithmetic operations. This is important because it allows for the simplification of code and the reduction of memory usage. It also allows for the reduction of the number of loops needed to perform the operation. When using broacasting the neural network is able to simplify the forward and backward passes, by allowing for the operation to be performed on tensors of different shapes. Another advantage is that it allows for batch processing of data, which is important for training neural networks, a 1D bias tensor can be added to a 2D tensor of weights.

### 3. Explain what limitations or challenges might arise when relying on broadcasting, particularly in the context of ensuring model correctness and efficiency.

- The first one that comes to my mind is when two tensors are incompatable shape wise. This can lead to errors in the code, and cause hours of debugging. Another thing that is a possibility when working on and with tensors is that there could be some unpredicable behavour that occurs.

**Example**

In [ ]:
```
# Tensor Broadcasting

tensor1 = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
tensor2 = torch.tensor([4, 5, 6])

'''
We have tensor 1 with shape (3, 3) and tensor 2 with shape (3,) and we are just goi
'''

result = tensor1 + tensor2


print("Ternsor 1: \n ", tensor1,"\n")

print("Tensor 2: \n ", tensor2,"\n")

print("Result: \n ", result,"\n")
```

```
Ternsor 1:
  tensor([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])

Tensor 2:
  tensor([4, 5, 6])

Result:
  tensor([[ 5,  7,  9],
        [ 8, 10, 12],
        [11, 13, 15]])
```