TENSORFLOW BASICS : Tensors

What is Tensor ?

- Tensors are primary data structure used for storing and manipulating the data in Deep Learning
- Example of Tensors:

01 **SCALAR**, Ex. 10.0

- Scalar is a rank 0 tensor, i.e with no axes
- Rank basically means how many axis a tensor has
- Shape = ( ), dtype = float32

What is Tensor ?

- Tensors are primary data structure used for storing and manipulating the data in Deep Learning
- Example of Tensors:

01 **SCALAR**, Ex. 10.0

- Scalar is a rank 0 tensor, i.e with no axes
- Rank basically means how many axis a tensor has
- Shape = ( ), dtype = float32

02 **VECTOR**, Ex. [1.0, 4.0, 7.0]

- Vector is a rank 1 tensor
- It has 1 axis, we can think like a row or column
- Shape = (3,), dtype = float32

What is Tensor ?

- Tensors are primary data structure used for storing and manipulating the data in Deep Learning
- Example of Tensors:

01	**SCALAR**, Ex. 10.0	• Scalar is a rank 0 tensor, i.e with no axes • Rank basically means how many axis a tensor has • Shape = ( ), dtype = float32
02	**VECTOR**, Ex. [1.0, 4.0, 7.0]	• Vector is a rank 1 tensor • It has 1 axis, we can think like a row or column • Shape = (3,), dtype = float32
03	**MATRIX**, Ex. [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]	• Matrix is a rank 2 tensor • It has 2 axes, we can think like row and column • Shape = (3, 3), dtype = float32

What is Tensor ?

- Tensors are primary data structure used for storing and manipulating the data in Deep Learning
- Example of Tensors:

01	**SCALAR**, Ex. 10.0	• Scalar is a rank 0 tensor, i.e with no axes • Rank basically means how many axis a tensor has • Shape = ( ), dtype = float32
02	**VECTOR**, Ex. [1.0, 4.0, 7.0]	• Vector is a rank 1 tensor • It has 1 axis, we can think like a row or column • Shape = (3,), dtype = float32
03	**MATRIX**, Ex. [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]	• Matrix is a rank 2 tensor • It has 2 axes, we can think like row and column • Shape = (3, 3), dtype = float32
04	**Higher-rank Tensor**	• An array with more than 2 axes, i.e. with 3, 4 etc.

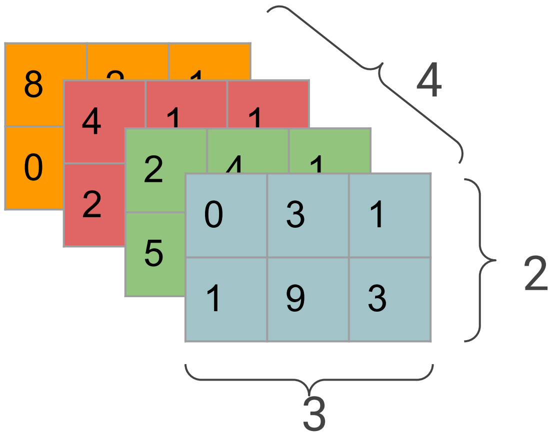
Definition: Tensors are multi dimensional array with all the elements of a Tensor having **<u>same data type</u>**.

Tensors are immutable, meaning once a tensor is created, its element cannot be changed. If any update is required then a new tensor needs to be created with the modification.

What is Tensor ? - Cont..

Let’s Visualize Rank 3 and Rank 4 Tensor, we will frequently encounter this when dealing with image, Textual and many other datasets and creating Batches out of it.

Let's take a Rank 3 tensor with shape = (4, 2, 3)



1 Shape

- Shape of a tensor gives information related to number of element along each axes

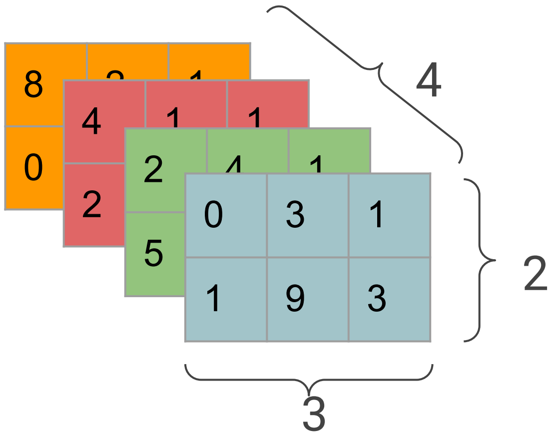
2 Size

- Size of a tensor is the total number of elements in the tensor, i.e. product of elements given by shape along each axes

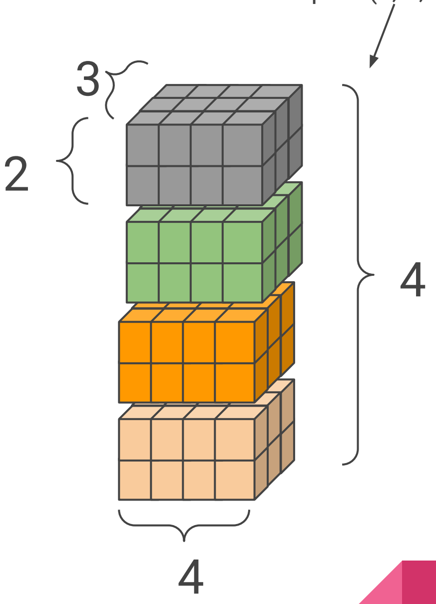
What is Tensor ? - Cont..

Let’s Visualize Rank 3 and Rank 4 Tensor, we will frequently encounter this when dealing with image, Textual and many other datasets and creating Batches out of it.

Let's take a Rank 3 tensor with shape = (4, 2, 3)



Let's take a Rank 4 tensor with shape = (4, 2, 4, 3)



We can also think of axis in terms of what it represent.

Axis of rank 4 tensor can be thought of as:

(Batch, Height, width, Features).

Ex: Think if example given as Batch of 4 images (2 * 4) with RGB Channel.

1	**Shape**	Shape of a tensor gives information related to number of element along each axes
2	**Size**	Size of a tensor is the total number of elements in the tensor, i.e. product of elements given by shape along each axes

Types of Tensors in TensorFlow - Constant Tensor

Some basic types of Tensors in TensorFlow: 1. Constant Tensor, 2. Variable Tensor, 3. Ragged Tensor, 4. Sparse Tensor

01 **Constant tensor**

- tf.constant() function can be used create a constant tensor
- It is immutable i.e. once created it cannot be changed
- Examples below

Types of Tensors in TensorFlow - Constant Tensor

Some basic types of Tensors in TensorFlow: 1. Constant Tensor, 2. Variable Tensor, 3. Ragged Tensor, 4. Sparse Tensor

01 Constant tensor

- tf.constant() function can be used create a constant tensor
- It is immutable i.e. once created it cannot be changed
- Examples below

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

```
[3]
constant = tf.constant([1.0,2.0,3.0])
print(constant)
# Numpy array can also be passed as an input to tf.constant
#shape and dtype is inferred from the input given if not given
array = np.ones((2,2))
constant_array = tf.constant(array)
print(constant_array)
#we can also further define shape parameter to reshape the constant tensor
constant_reshaped = tf.constant(array,shape=(1,4))
print(constant_reshaped)
```

```
tf.Tensor([1. 2. 3.], shape=(3,), dtype=float32)
tf.Tensor(
[[1. 1.]
 [1. 1.]], shape=(2, 2), dtype=float64)
tf.Tensor([[1. 1. 1. 1.]], shape=(1, 4), dtype=float64)
```

```
#Constant tensor cannot be modified. we will get error while trying to do so
try:
    constant.assign([4.0,5.0,6.0])
except AttributeError as e:
    print(e)
```

```
'tensorflow.python.framework.ops.EagerTensor' object has no attribute 'assign'
```

```
# Constant tensor can be exported to numpy as well
constant_to_numpy = constant.numpy()
constant_to_numpy
```

```
array([1., 2., 3.], dtype=float32)
```

Types of Tensors in TensorFlow - Variable Tensor

02

Variable tensor

- tf.Variable() function can be used create a variable tensor and like constant tensor can be exported to numpy
- It is mutable, i.e. It's value can be changed even after it is created, but cannot be reshaped
- As it is mutable, variable tensors are used to store models parameters like weights and biases, as these are regularly updated during training

```
tf.Variable( initial_value=None, trainable=None, name=None, dtype=None)
```

Types of Tensors in TensorFlow - Variable Tensor

02

Variable tensor

- tf.Variable() function can be used create a variable tensor and like constant tensor can be exported to numpy
- It is mutable, i.e. It's value can be changed even after it is created, but cannot be reshaped
- As it is mutable, variable tensors are used to store models parameters like weights and biases, as these are regularly updated during training

tf.Variable(initial_value=None, trainable=None, name=None, dtype=None)

Initial_value:

- Initial value that will be assigned to the variable. It can be a tensor or any python object like numpy array that can be converted to a Tensor.
- Data type and shape of the tensor is automatically inferred based on initial_value

```
[7]
variable = tf.Variable([2,4,6])
print(variable)
var_arr = np.ones((2,2))
variable_array = tf.Variable(var_arr)
print(variable_array)
```

```
<tf.Variable 'Variable:0' shape=(3,) dtype=int32, numpy=array([2, 4, 6], dtype=int32)>
<tf.Variable 'Variable:0' shape=(2, 2) dtype=float64, numpy=
array([[1., 1.],
       [1., 1.]])>
```

Types of Tensors in TensorFlow - Variable Tensor

tf.Variable(initial_value=None, trainable=None, name=None, dtype=None)

trainable : Takes boolean True or False, by default takes as True.

- setting the trainable parameter to true ensures the variable is watched by tensorflow.

Intuitively, we can think of it like tensorflow is monitoring and keeping notes of operations or chain of operations related to the variables like Weights and Bias of a model during model training.

For ex., where in the computation the specific variable is used, how and when is it getting updated, how it interact with other variables and operation in model.

Keeping track or log of the variables life-cycle enables tensorflow to handles many complex computation internally like Calculating gradient with respect to variable and updating it based on that, variable tracking is also leveraged while saving or creating checkpoint of model with current variable values.

Types of Tensors in TensorFlow - Variable Tensor

tf.Variable(initial_value=None, trainable=None, name=None, dtype=None)

trainable : Takes boolean True or False, by default takes as True.

- setting the trainable parameter to true ensures the variable is watched by tensorflow.

Intuitively, we can think of it like tensorflow is monitoring and keeping notes of operations or chain of operations related to the variables like Weights and Bias of a model during model training.

For ex., where in the computation the specific variable is used, how and when is it getting updated, how it interact with other variables and operation in model.

Keeping track or log of the variables life-cycle enables tensorflow to handles many complex computation internally like Calculating gradient with respect to variable and updating it based on that, variable tracking is also leveraged while saving or creating checkpoint of model with current variable values.

[5]

```
train_var = tf.Variable(2.0,trainable=True)
no_train_var = tf.Variable(3.0,trainable=False)
# GradientTape is the tensorflow API used to calcilate gradient w.r.t variable during model training
# Will understand this API in more detail in later videos
with tf.GradientTape(persistent = True) as tape:
  # Defining chain of equation w.r.t variables defined
  z1 = train_var * no_train_var
  z2 = z1**2
# Since no_train_var trainable parameter is set to False,
# tensorflow will not watch this variable life-cycle and hence will not give gradient
print(tape.gradient(z2,no_train_var))
# expected answer dz2/dtrain_var = 2*z1*no_train_var = 2.0*(2.0*3.0)*3.0 = 36.0 -> Simple differentiation
print(tape.gradient(z2,train_var))
```

```
None
tf.Tensor(36.0, shape=(), dtype=float32)
```

Types of Tensors in TensorFlow - Variable Tensor

tf.Variable(initial_value=None, trainable=None, name=None, dtype=None)

name:

- We can use name to give specific name to variable. By default it take name as 'Variable'. We can give same name to two variables as well.

dtype:

- If dtype is mentioned then initial value will be converted to mentioned dtype.

```python
default_name = tf.Variable([2.0])
print(default_name)
#Giving a unique name to variable
variable_name = tf.Variable([5.0,6.0,7.0],name='variable_new_name')
print(variable_name)
# Explicit dtype
variable_dtype = tf.Variable([2,3,4],dtype=tf.float64)
print(variable_dtype)

# We should not give incompatible shape parameter while creating variable
try:
  var_incompatible_shape = tf.Variable([2,3,4],shape=(2,3))
except ValueError as e:
  print(e)
```

```
<tf.Variable 'Variable:0' shape=(1,) dtype=float32, numpy=array([2.], dtype=float32)>
<tf.Variable 'variable_new_name:0' shape=(3,) dtype=float32, numpy=array([5., 6., 7.], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float64, numpy=array([2., 3., 4.])>
In this `tf.Variable` creation, the initial value's shape ((3,)) is not compatible with the explicitly supplied `shape` argument ((2, 3)).
```

Types of Tensors in TensorFlow - Variable Tensor

Exporting variable Tensor to numpy as well as accessing shape, dtype of a variable Tensor

```
var = tf.Variable([[1,2],[3,4]])
print(var)
print(var.shape)
print(var.dtype)
print(var.numpy())
```

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

Types of Tensors in TensorFlow - Variable Tensor

Exporting variable Tensor to numpy as well as accessing shape, dtype of a variable Tensor

```
var = tf.Variable([[1,2],[3,4]])
print(var)
print(var.shape)
print(var.dtype)
print(var.numpy())
```

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

We can change the value of variable tensor by reassign the tensor using **tf.Variable.assign,** using assign generally uses same tensor memory, but when we create a new variable using existing variable both variable will have different memory

```
var1 = tf.Variable([3,4,5])
var2 = tf.Variable(var1)
var1.assign([9,1,2])
print(var1)
print(var2)
# We cannot assign value of different shape and type
try:
    var1.assign([1,2,3,4])
except ValueError as e:
    print(e)
```

```
<tf.Variable 'Variable:0' shape=(3,) dtype=int32, numpy=array([9, 1, 2], dtype=int32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=int32, numpy=array([3, 4, 5], dtype=int32)>
Cannot assign value to variable ' Variable:0': Shape mismatch.The variable shape (3,), and the assigned value shape (4,) are incompatible.
```

```
try:
    var1.assign([1.0,2.0,3.0])
except TypeError as e:
    print(e)
```

```
Cannot convert [1.0, 2.0, 3.0] to EagerTensor of dtype int32
```

Types of Tensors in TensorFlow - Variable Tensor

Other ways to assign value to variable by adding or subtracting delta from current value

```python
var2 = tf.Variable([4.0,5.0,1.0])
var2.assign_add([4.0,1.0,2.0])
print(var2)
var2.assign_sub([7.0,9.0,6.0])
print(var2)
```

```
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([8., 6., 3.], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([ 1., -3., -3.], dtype=float32)>
```

Types of Tensors in TensorFlow - Ragged Tensor

Some basic types of Tensors in TensorFlow

03

Ragged tensor

- Regular Tensors needs to be rectangular, i.e. no. of elements along the axis should remain same
- So, Tensor with variable number of elements along some axis is called Ragged Tensor
- Simplest way to create Ragged Tensor is to use tf.ragged.constant()
- All the element should have same data type as well as same nesting depth.

Types of Tensors in TensorFlow - Ragged Tensor

Some basic types of Tensors in TensorFlow

03 **Ragged tensor**

- Regular Tensors needs to be rectangular, i.e. no. of elements along the axis should remain same
- So, Tensor with variable number of elements along some axis is called Ragged Tensor
- Simplest way to create Ragged Tensor is to use tf.ragged.constant()
- All the element should have same data type as well as same nesting depth.

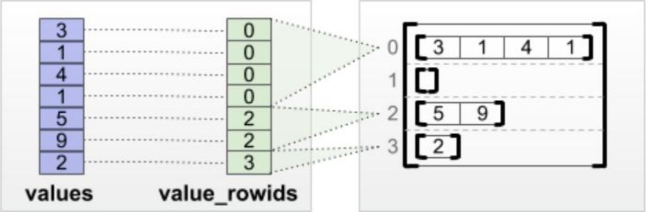
```
non_rect_arr = [ [1.0,2.0,3.0],
                 [5.0,7.0],
                 [9.0]
               ]
try:
    non_rect_const = tf.constant(non_rect_arr)
except ValueError as e:
    print(e)
ragged_tensor = tf.ragged.constant(non_rect_arr)
print(ragged_tensor)
print(ragged_tensor.shape)
```

```
Can't convert non-rectangular Python sequence to Tensor.
<tf.RaggedTensor [[1.0, 2.0, 3.0], [5.0, 7.0], [9.0]]>
(3, None)
```

Types of Tensors in TensorFlow - Ragged Tensor

Let's see few other ways to create ragged tensor:

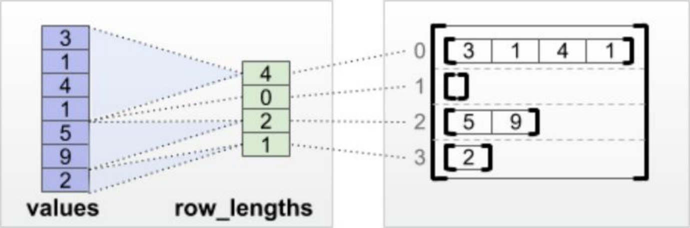
1. tf.RaggedTensor.from_value_rowids




```python
ragged1 = tf.RaggedTensor.from_value_rowids(
    values=[3, 1, 4, 1, 5, 9, 2],
    value_rowids=[0, 0, 0, 0, 2, 2, 3])
print(ragged1)
```

```
<tf.RaggedTensor [[3, 1, 4, 1], [], [5, 9], [2]]>
```

2. tf.RaggedTensor.from_row_lengths




```python
ragged2 = tf.RaggedTensor.from_row_lengths(
    values=[3, 1, 4, 1, 5, 9, 2],
    row_lengths=[4, 0, 2, 1]
    )
print(ragged2)
```

```
<tf.RaggedTensor [[3, 1, 4, 1], [], [5, 9], [2]]>
```

Types of Tensors in TensorFlow - Sparse Tensor

04

Sparse tensor

- Sometime the tensor we want to work with will have lot of zeros in it, which is referred to as sparse tensor
- In such case the more memory efficient way to store the tensor would be to store only non-zero values with their coordinate rather than complete tensor

Types of Tensors in TensorFlow - Sparse Tensor

04

Sparse tensor

- Sometime the tensor we want to work with will have lot of zeros in it, which is referred to as sparse tensor
- In such case the more memory efficient way to store the tensor would be to store only non-zero values with their coordinate rather than complete tensor

```python
'''
Consider below example where most of the entries is 0. If we can store only
non-zero values and its coordiante it will be more efficient than storing
complete tensor. tf.sparse.SparseTesnor is used to store exactly these kind
of sparse tensor.

[[1,0,0,0],
  [0,2,0,0],
  [0,0,0,0],
  [0,0,0,0]]

'''
sparse_tensor = tf.sparse.SparseTensor(indices = [[0,0],[1,1]],values = [1,2],dense_shape = [4,4])
print(sparse_tensor)
# We can convert sparse tensor to dense tensor
print(tf.sparse.to_dense(sparse_tensor))
```

```
SparseTensor(indices=tf.Tensor(
[[0 0]
 [1 1]], shape=(2, 2), dtype=int64), values=tf.Tensor([1 2], shape=(2,), dtype=int32), dense_shape=tf.Tensor([4 4], shape=(2,), dtype=int64))
tf.Tensor(
[[1 0 0 0]
 [0 2 0 0]
 [0 0 0 0]
 [0 0 0 0]], shape=(4, 4), dtype=int32)
```

Indexing

- Indexing is very similar to indexing rule followed for python list or numpy array
- Single Axis:
    - Index starts at 0
    - negative index counts from end
    - start:stop:step with colon is used to slice
    - using colon keeps the axis
- For Multi-Axis indexing for higher rank tensor we apply single axis rule to each axis independently separated by comma

Indexing

- Indexing is very similar to indexing rule followed for python list or numpy array
- Single Axis:
  - Index starts at 0
  - negative index counts from end
  - start:stop:step with colon is used to slice
  - using colon keeps the axis
- For Multi-Axis indexing for higher rank tensor we apply single axis rule to each axis independently separated by comma

```
# Single axis Tensor
single_axis_Tensor = tf.constant([1,2,3,4,5,6,7,8,9])
print(single_axis_Tensor)
# Axis will be gone
print(single_axis_Tensor[3])
# We will keep axis using :
print(single_axis_Tensor[2:3])
# Select from begining to end but take every second element
print(single_axis_Tensor[0:11:2])
# print in reverse order using negative index
print(single_axis_Tensor[::-1])
```

```
tf.Tensor([1 2 3 4 5 6 7 8 9], shape=(9,), dtype=int32)
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor([3], shape=(1,), dtype=int32)
tf.Tensor([1 3 5 7 9], shape=(5,), dtype=int32)
tf.Tensor([9 8 7 6 5 4 3 2 1], shape=(9,), dtype=int32)
```

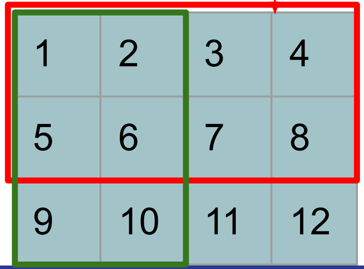
Indexing

- Multi-Axis indexing example and visualization

```python
# Multi Axis Tensor
multi_axis_tensor1 = tf.constant([
        [1,2,3,4],
        [5,6,7,8],
        [9,10,11,12]
        ])
print(multi_axis_tensor1)
print(multi_axis_tensor1[:-1,0:2])
```

```
tf.Tensor(
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]], shape=(3, 4), dtype=int32)
tf.Tensor(
[[1 2]
 [5 6]], shape=(2, 2), dtype=int32)
```

This indexing means along rows take all but last row (: -1 ,) and along column take 1st and 2nd column (, 0:2)



Indexing

- Multi-Axis indexing example and visualization

```
# Multi Axis Tensor
multi_axis_tensor1 = tf.constant([
        [1,2,3,4],
        [5,6,7,8],
        [9,10,11,12]
        ])
print(multi_axis_tensor1)
print(multi_axis_tensor1[:-1,0:2])
```

```
tf.Tensor(
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]], shape=(3, 4), dtype=int32)
tf.Tensor(
[[1 2]
 [5 6]], shape=(2, 2), dtype=int32)
```

```
multi_axis_tensor2 = tf.constant([
        [[1,2,3],[4,5,6]],
        [[7,8,9],[10,11,12]],
        [[13,14,15],[16,17,18]],
        [[19,20,21],[22,23,24]]
        ])
print(multi_axis_tensor2)
```

```
tf.Tensor(
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[13 14 15]
  [16 17 18]]

 [[19 20 21]
  [22 23 24]]], shape=(4, 2, 3), dtype=int32)
```

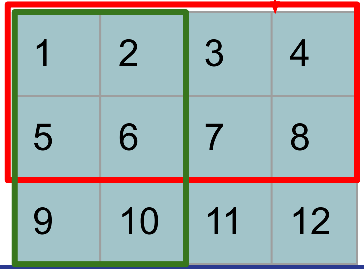
```
print(multi_axis_tensor2[:,:,0])
```

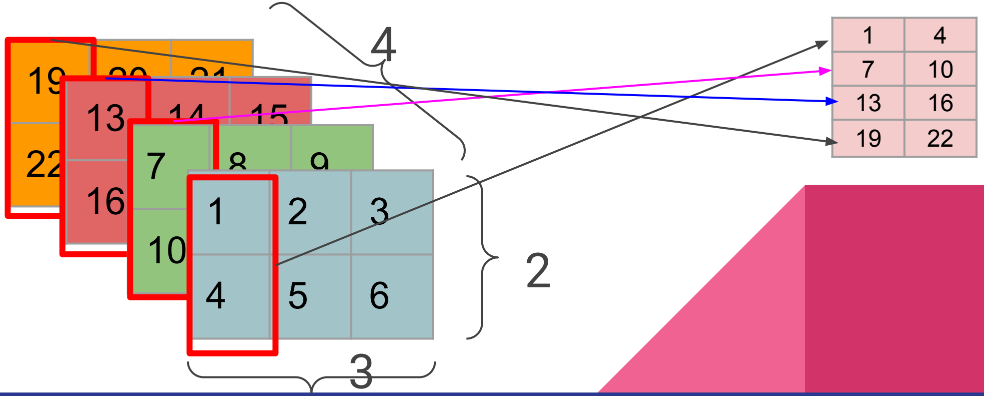
```
tf.Tensor(
[[ 1  4]
 [ 7 10]
 [13 16]
 [19 22]], shape=(4, 2), dtype=int32)
```

This indexing intuitively means that take first column denoted by 0 in the index for each example across the batch denote by [: , : ,]

Alternatively we can say we are selecting first feature across all location for each example in the batch

This indexing means along rows take all but last row (: -1 ,) and along column take 1st and 2nd column (, 0:2)





Reshaping Tensors

There are many scenarios where we may need to reshape our tensors. **tf.reshape()** help us do that easily. Few examples below

Reshaping Tensors

```
constant_tensor = tf.constant([1,2,3,4,5,6,7,8,9,0])
print(constant_tensor)
print(tf.reshape(constant_tensor,[5,2]))
```

```
tf.Tensor([1 2 3 4 5 6 7 8 9 0], shape=(10,), dtype=int32)
tf.Tensor(
[[1 2]
 [3 4]
 [5 6]
 [7 8]
 [9 0]], shape=(5, 2), dtype=int32)
```

[17]

```
#flatten the tensor
constant_tesnor_1 = tf.constant([[1,2,3],[4,5,6]])
print(constant_tesnor_1)
print(tf.reshape(constant_tesnor_1,[-1]))
```

```
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)
tf.Tensor([1 2 3 4 5 6], shape=(6,), dtype=int32)
```

```
#combining adjacent axis
print(multi_axis_tensor2)
print(tf.reshape(multi_axis_tensor2,[4*2,-1]))
```

```
tf.Tensor(
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]

 [[13 14 15]
  [16 17 18]]

 [[19 20 21]
  [22 23 24]]], shape=(4, 2, 3), dtype=int32)
tf.Tensor(
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]
 [13 14 15]
 [16 17 18]
 [19 20 21]
 [22 23 24]], shape=(8, 3), dtype=int32)
```

Thank You