Tf.constant()

Tf.variable()

Tensor[0].assign(8)

**Random Tensors**

random1 = tf.random.Generator.from\_seed(42)

random1 = random1.normal(shape=(3,2))

normal distribution or Uniform distribution

**Shuffle**:

# Shuffle in the same order every time

# Set the global random seed

tf.random.set\_seed(42)

# Set the operation random seed

tf.random.shuffle(not\_shuffled, seed=42)

Global vs local or operational random seed

**Numpy to tensors**

numpy\_A = np.arange(1, 25, dtype=np.int32) # create a NumPy array between 1 and 25

A = tf.constant(numpy\_A,

shape=[2, 4, 3])

tf.ones([10,10])

tf.zeros(shape=(10,11))

Tensor info

random\_tensor.shape[0],random\_tensor.ndim , tf.size(random\_tensor).numpy()

---Deepmind.com

changeable\_tensor = tf.Variable([10, 7])

unchangeable\_tensor = tf.constant([10, 7])

changeable\_tensor[0].assign(7)

random\_1 = tf.random.Generator.from\_seed(42) # set the seed for reproducibility

random\_1 = random\_1.normal(shape=(3, 2))

Shuffling: tf.random.shuffle(not\_shuffled)

tf.random.shuffle(not\_shuffled, seed=42)

# Set the global random seed

tf.random.set\_seed(42)

# Set the operation random seed

tf.random.shuffle(not\_shuffled, seed=42)

rank\_4\_tensor = tf.zeros([2, 3, 4, 5])

**Indexing**

# Get the first 2 items of each dimension

rank\_4\_tensor[:2, :2, :2, :2]

**Expanding**

Add axis:: rank\_3\_tensor = rank\_2\_tensor[..., tf.newaxis]

tf.expand\_dims(rank\_2\_tensor, axis=-1) # "-1" means last axis

**Basic operation**

tensor = tf.constant([[10, 7], [3, 4]])

tensor + 10

tf.multiply(tensor, 10)

**# Matrix multiplication with Python operator '@'**

tensor @ tensor

tf.matmul(tensor, tensor)

# Example of reshape (3, 2) -> (2, 3)

tf.reshape(Y, shape=(2, 3))

X @ tf.reshape(Y, shape=(2, 3))

# Example of transpose (3, 2) -> (2, 3)

tf.transpose(X)

# Try matrix multiplication

tf.matmul(tf.transpose(X), Y)

# You can achieve the same result with parameters

tf.matmul(a=X, b=Y, transpose\_a=True, transpose\_b=False)

# Perform the dot product on X and Y (requires X to be transposed)

tf.tensordot(tf.transpose(X), Y, axes=1)

**Casting**

# Change from float32 to float16 (reduced precision)

B = tf.cast(B, dtype=tf.float16)

B

**Aggregation**

**Finding the min, max, mean, sum (aggregation)**

**E = tf.constant(np.random.randint(low=0, high=100, size=50))**

**tf.reduce\_min(E)**

**tf.reduce\_max(E)**

**tf.reduce\_mean(E)**

Variance and Standard Deviation

import tensorflow\_probability as tfp

tfp.stats.variance(B)

tfp.stats.stddev(tf.cast(B,dtype=tf.float32))

tf.math.reduce\_std(tf.cast(B,dtype=tf.float32))

tf.math.reduce\_variance(tf.cast(B,dtype=tf.float32))

**Postional Maximum and Minimum**

F = tf.random.uniform([50])

F[tf.argmax(F)]

assert F[tf.argmax(F)] == tf.reduce\_max(F)

**Squeezing**

tf.random.set\_seed(42)

G = tf.constant(tf.random.uniform(shape=[50]),shape=(1,1,1,1,50))

G

G\_squeezed = tf.squeeze(G)

G\_squeezed , G

**Onehot encoding**

tf.one\_hot(some\_list,depth=4)

tf.one\_hot(some\_list,depth=4,on\_value="ON",off\_value="off")

**Squaring, log, square root**

H = tf.range(0,10)

tf.square(H)

tf.sqrt(tf.cast(H,dtype=tf.float32))

tf.math.log(tf.cast(H,dtype=tf.float32))

**Tensors and Numpy**

A = tf.constant(np.array([1,2,3]))

np.array(A), type(np.array(A))

A.numpy(), type(A.numpy())

B = tf.constant(np.array([1,2,3]))

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

B.dtype, C.dtype

**Np.array is of 64 bit**