



TensorFlow Fundamentals

Table of contents:

Introduction to Tensors

Create tensors using `tf.constant`

Create tensors using `tf.Variable`

Why can we change some tensors and why we can't change some tensors ? 🤔

Creating random tensors

Why do we want to create tensors with random numbers ?

Creating random tensors using `tf.random`

Shuffling the order of tensors

Why do we want reproducible tensors ?

Other ways to make tensors

Turn NumPy arrays into tensors

Getting information from tensors

Indexing tensors

Expanding tensors

Manipulating tensors (tensor operations)

Basic Operations

TensorFlow built-in math functions

Matrix Multiplication

Matrix Multiplication in TensorFlow

Matrix Multiplication with tensors of different shapes

Tensor transformations - transpose

The dot product

Changing the datatypes of tensors

Aggregating tensors

Find the position minimum and maximum

Why do we need to find position maximum and position minimum ? 🤔

`tf.argmax()`

`tf.argmin()`

Squeezing a tensor (removing all single dimensions)


One-hot Encoding tensors

More Math Operations

Tensors and NumPy

Colab Notebook for this section:

Google Colaboratory

 <https://colab.research.google.com/drive/1S639pbVUKWBI-O4FBtILNjh1e2XI1D3E#scrollTo=rcWQKmV7I7ES>



TensorFlow Documentation:

Module: tf | TensorFlow Core v2.8.0

TensorFlow

 https://www.tensorflow.org/api_docs/python/tf

Introduction to Tensors

As we have covered, tensor - is a numerical way to represent some information.

There are few ways to create tensors. In general, we won't really create many tensors ourselves. This is because TensorFlow has many built-in modules.


We are going to cover 2 ways to create tensors:

- Using `tf.constant`
- Using `tf.Variable`

Create tensors using `tf.constant`

`tf.constant` | TensorFlow Core v2.7.0

Creates a constant tensor from a tensor-like object.

 https://www.tensorflow.org/api_docs/python/tf/constant?hl=ru

We are going to create the following:

Name	Definition	Number of dimensions
Scalar	Single number	0
Vector	Number with direction (e.g wind speed and direction)	1
Matrix	2-dimensional array of numbers	2
Tensor	N-dimensional array of numbers (where n can be any number)	N

```
# Create a scalar using tf.constant
scalar = tf.constant(7)
scalar
```

```
# Create a vector, passing Python List
vector = tf.constant([10,10])
vector
```

```
# Create a matrix (has more than 1 dimension)
matrix = tf.constant([[10,7],
                      [7,10]])
matrix
```

We can specify the data type using `dtype` parameter.

```
# Create another matrix
another_matrix = tf.constant([[10., 7.],
                              [3., 2.],
                              [8.,9.]], dtype=tf.float16) # specify the data type with dtype parameter
another_matrix
```

```
# Create a tensor
tensor = tf.constant([[[1,2,3],
                       [4,5,6]],
                      [[7,8,9],
                       [10,11,12]],
                      [[13,14,15],
                       [16,16,18]]])
tensor
```

We can check the number of dimensions using `ndim` property.

```
# What is the number of dimensions of the tensor ?
tensor.ndim
```

Create tensors using `tf.Variable`

tf.Variable | TensorFlow Core v2.7.0

See the [variable guide](<https://tensorflow.org/guide/variable>).

 https://www.tensorflow.org/api_docs/python/tf/Variable

- The `tf.Variable()` constructor requires an initial value for the variable, which can be a Tensor of any type and shape.
- This initial value defines the type and shape of the variable.
- After construction, the type and shape of the variable are fixed.
- The **value can be changed** using one of the assign methods.

```
# Create tensor with tf.Variable()
changeable_tensor = tf.Variable([10,7])
# Create tensor with tf.constant
unchangeable_tensor = tf.constant([10,7])

changeable_tensor, unchangeable_tensor
```

Tensor, which is created with `tf.Variable`, is **changeable** using `assign()` method.

```
changeable_tensor[0].assign(7)
changeable_tensor
```

Tensor, which is created with `tf.constant` is **unchangeable**.

Why can we change some tensors and why we can't change some tensors ? 🤔

When we are writing new Neural Network, we might want some tensors values **to be changed** (`tf.Variable`) and we might want some tensors values **not to be changed** (`tf.constant`).



Note: Rarely in practice will you need to decide whether to use `tf.constant` or `tf.variable` to create tensors, as TensorFlow does this for you. However, if in doubt, use `tf.constant` and change it later if needed.

Creating random tensors

Random tensors are tensors of some arbitrary size which contain random numbers.

Why do we want to create tensors with random numbers ?

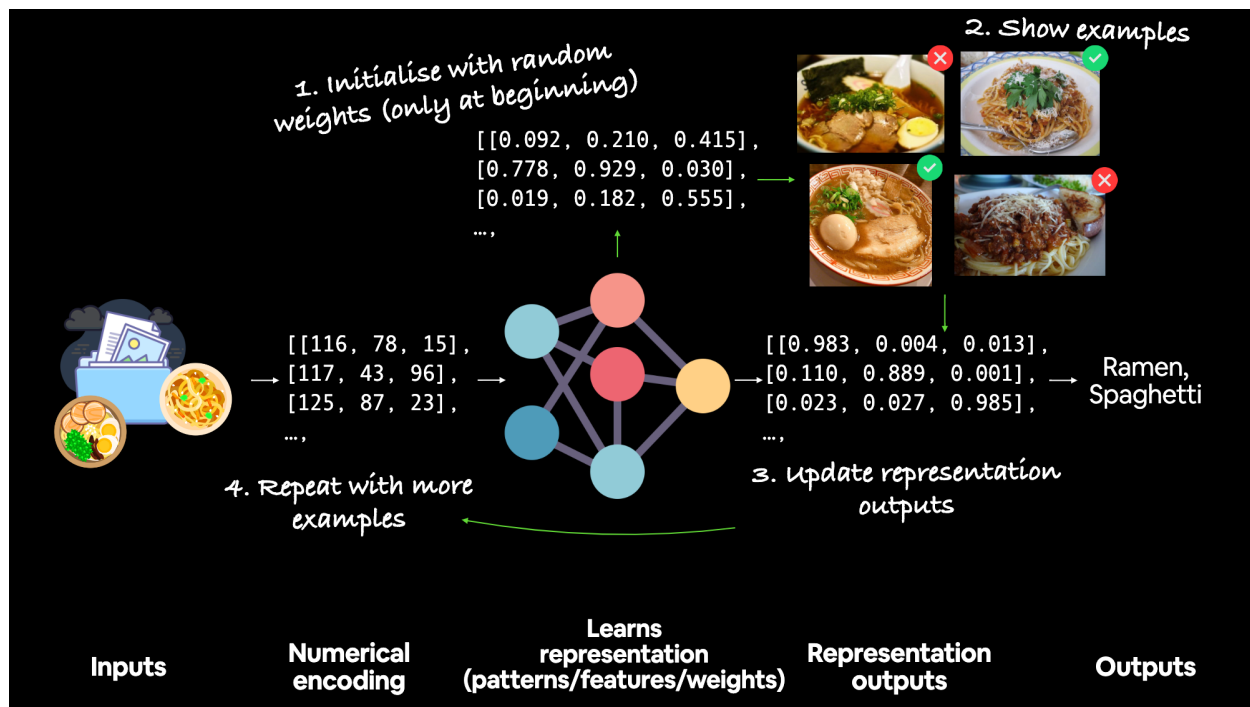


Diagram description:

Inputs: As our inputs we have pictures of food that we are trying to classify into Ramen or Spaghetti.

Numerical encoding: We then turn those pictures into tensors (some numerical way to represent information)

Learns representation (patterns/features/weights): Then, we pass our tensors to the input layers of our Neural Network. Our Neural Network might learn representations which are called patterns, features, weights.

Representation outputs: Then, our Neural Network outputs those representation outputs.

Outputs: Finally, we convert representation outputs into something that we can understand rather than just tensors.

1. This is what Neural Networks use to initialize their weights (in other words patterns) that they are trying to learn in our data. **In the beginning, our Neural Network is going to initialize its weights or the patterns it knows with random weights or random tensors.**
2. Then, as it sees different examples of photos (food images), it is going to update its representation outputs.

3. In other words, it is going to tweak these random weights/patterns to better suit our data.
4. We are going to repeat this cycle with more and more examples.

So, when our Neural Network starts to learn, it wants to learn patterns in some sort of dataset. It starts off with random patterns. Then, it slowly adjusts them as it continually learns on more and more examples.

Creating random tensors using `tf.random`

We can create random tensors using `tf.random`.

```
# Create two random (but the same) tensors
random_1 = tf.random.Generator.from_seed(42) # set seed for reproducibility
random_1 = random_1.normal(shape=(3,2))
random_1

random_2 = tf.random.Generator.from_seed(42)
random_2 = random_2.normal(shape=(3,2))
random_2
```

The following two tensors will be equal to each other.

Q: Why are they both equal ? 🤔

A: Those random tensors are actually **pseudo random numbers**. They appear as random, but they are really aren't. That's because we have set the seed using `from_seed()` method.

Shuffling the order of tensors

Let say we are working on food classification. We have 15000 images of ramen and spaghetti:

1. The first 10000 images of ramen
2. The last 5000 images of spaghetti


This order could affect how our Neural Network works. So, if it goes through these images in order, it might start to adjust its random weights too much. If it goes through 10000 images of ramen, it will learn only what ramen looks like. It doesn't know that it has to also learn what spaghetti looks like until it goes to the last 5000 images of spaghetti.

Instead it might be a good idea to mix up all the images in our folder so that they basically have no order at all. Then our Neural Network can be fed different examples of different images and adjust its internal patterns/weights (random tensors that it got initialized with) to learn both types of images at the same time.

To shuffle a tensor, use `tf.random.shuffle`

`tf.random.shuffle` | TensorFlow Core v2.7.0

Randomly shuffles a tensor along its first dimension.

 https://www.tensorflow.org/api_docs/python/tf/random/shuffle

The tensor is shuffled along dimension 0, such that each `value[j]` is mapped to one and only one `output[i]`

This code implementation will get tensor randomly shuffled each time along dimension 0.

```
# Shuffle a tensor (valuable for when you want to shuffle your data so the inherent order doesn't affect learning)
not_shuffled = tf.constant([[10, 7],
                             [3, 4],
                             [2, 5]])

not_shuffled

# Shuffle our non-shuffled tensor
tf.random.shuffle(not_shuffled)
```

We can set a seed to get the same order. `tf.random.set_seed`

`tf.random.set_seed` | TensorFlow Core v2.7.0

Sets the global random seed.

 https://www.tensorflow.org/api_docs/python/tf/random/set_seed

```
# Shuffle our non-shuffled tensor with seed
tf.random.set_seed(42) # Global level random seed
tf.random.shuffle(not_shuffled, seed=42) # Operation level random seed
```

If we want our shuffled tensors to be in the same order, we have got to use the global level random seed as well as operation level random seed.

Rule 4: If both the global and the operation seed are set: Both seeds are used in conjunction to determine the random sequence.

https://www.tensorflow.org/api_docs/python/tf/random/set_seed

Why do we want reproducible tensors ?

As we start run more Deep Learning experiments, we will often find is that because a Neural Network initializes itself with random patterns/weights, we could get different results every single time we run the experiment.


So, to make reproducible experiments, we probably want to shuffle the data in a similar order, initialize with the similar random pattern/weights and then run through the experiment.

So, if we want to set the random seed, TensorFlow has a few rules that we need to follow:

1. If neither the global seed nor the operation seed is set: A randomly picked seed is used for this op.
2. If the global seed is set, but the operation seed is not: The system deterministically picks an operation seed in conjunction with the global seed so that it gets a unique random sequence. Within the same version of tensorflow and user code, this sequence is deterministic. However across different versions, this sequence might change. If the code depends on particular seeds to work, specify both global and operation-level seeds explicitly.
3. If the operation seed is set, but the global seed is not set: A default global seed and the specified operation seed are used to determine the random sequence
4. If both the global and the operation seed are set: Both seeds are used in conjunction to determine the random sequence.

tf.random.set_seed | TensorFlow Core v2.7.0

Sets the global random seed.

 https://www.tensorflow.org/api_docs/python/tf/random/set_seed

Other ways to make tensors

We can:

- generate a tensor with all elements set to one. It is similar to `Numpy.ones`

```
# Create a tensor of all ones, shape = 10, 7
tf.ones([10,7])
```

- generate a tensor with all elements set to zero. It is similar to `Numpy.zeros`

```
# Create a tensor of all zeros
tf.zeros(shape=(10,7))
```


Turn NumPy arrays into tensors

The main difference between NumPy arrays and TensorFlow tensors is that tensors can be run on a GPU (much faster for numerical computing). Otherwise, they are really similar.

```
import numpy as np

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

numpy_A
# X = tf.constant(some_matrix) # capital for matrix or tensor
# y = tf.constant (vector) # non-capital for vector

# Convert NumPy array into Tensors form
A = tf.constant(numpy_A) # This is a vector
# Changing the shape of it
B = tf.constant(numpy_A, shape=(2,3,4)) # This is a tensor, it has got more than 1 dimension
A, B
```

We can change the shape of the tensor.



Note: Important thing to know about shape is that if we want to readjust the shape of a tensor or an array, the new elements(values) in **shape** must add up to get the same number of elements in the original array/tensor

Getting information from tensors

Most important tensor terms/attributes:

Attribute	Meaning	Code
Shape	The length (number of elements) of each of the dimensions of a tensor	tensor.shape
Rank	The number of tensor dimensions. A scalar has rank 0, a vector has rank 1, a matrix has rank 2, a tensor has rank n	tensor.ndim
Axis or dimension	A particular dimension of a tensor	tensor[0], tensor[:, 1]
Size	The total number of items in the tensor	tf.size(tensor)

```
# Get various attributes of our tensor
print("Datatype of every element: ", rank_4_tensor.dtype)
print("Number of dimensions (rank): ", rank_4_tensor.ndim)
print("Shape of tensor: ", rank_4_tensor.shape)
print("Elements along the 0 axis", rank_4_tensor.shape[0])
print("Element along the last axis: ", rank_4_tensor.shape[-1])
print("Total number of elements in our tensor: ", tf.size(rank_4_tensor).numpy())
```

Indexing tensors

Tensors can be indexed just like Python list.

```
# Get the first 2 elements of each dimension
rank_4_tensor[:2, :2, :2, :2]

# Get the first element from each dimension from each index except for the final one
rank_4_tensor[:,1,1,1,:] # single column - get all elements

# Create a rank 2 tensor (2 dimensions)
rank_2_tensor = tf.constant([[10,7],
                             [3,4]])
rank_2_tensor.shape, rank_2_tensor.ndim
# Get the last item of each of our rank 2 tensor
rank_2_tensor[:, -1]
```

Expanding tensors

Ways to expand tensors:

- `tf.newaxis`
- `expand_dims(...)`

```
# Add in extra dimension to our rank 2 tensor
rank_3_tensor = rank_2_tensor[..., tf.newaxis]
rank_3_tensor

# Alternative to tf.newaxis
tf.expand_dims(rank_2_tensor, axis=-1) # "-1" means expand the final axis





tf.expand_dims(rank_2_tensor, axis=0) # expand the 0-axis
```

Manipulating tensors (tensor operations)

Finding patterns in data that is stored in tensor format, often involves manipulating tensors.

When building deep learning models, much of the tensor manipulation is done for us.

Basic Operations

-  addition
-  subtraction
-  multiplication
-  division

Basic operations are often referred to as **element wise operations**. Element wise means go through one element at a time and apply math operation (addition, subtraction, multiplication, division)

```
# We can add values to a tensor using the addition operator
tensor = tf.constant([[10,7], [3,4]])
tensor + 10
```


The original tensor remains unchanged when applying math operators.

```
# Multiplication also works
tensor * 10
# Subtraction
tensor - 10
# Division
tensor / 10
```

TensorFlow built-in math functions

Module: tf.math | TensorFlow Core v2.8.0

Math Operations.

 https://www.tensorflow.org/api_docs/python/tf/math

We can use the tensorflow built-in functions to manipulate tensors too.



Note: If we have to do some math tensorflow operations and want our code to be speed up (execute faster) on GPU, it is preferred to use tensorflow built-in functions.

```
tf.multiply(tensor, 10)
```

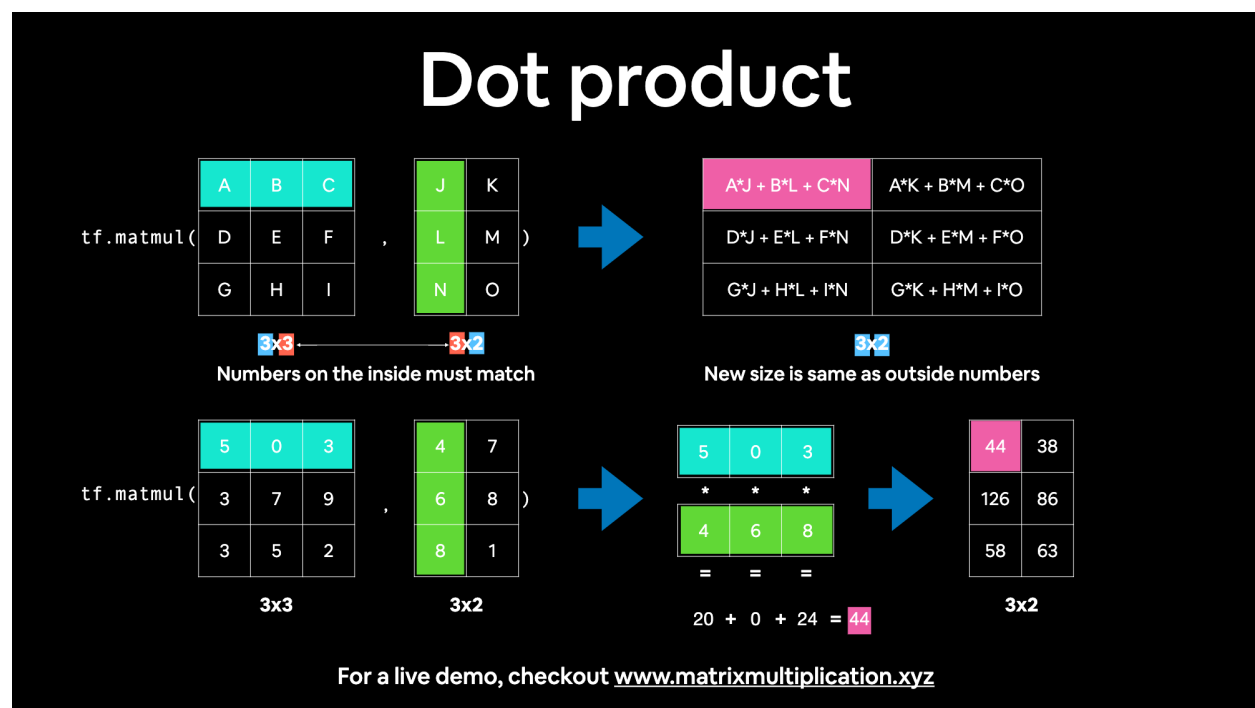
The original tensor remains unchanged when applying tensorflow math built-in functions

Matrix Multiplication

In Machine Learning, matrix multiplication is one of the most common tensor operations.

There are two rules our tensors (or matrices) need to fulfil if we're going to matrix multiply them:

1. The inner dimensions must match
2. The resulting has the shape of the outer dimensions




Matrix Multiplication in TensorFlow

We can do matrix multiplication in TensorFlow by using `tf.linalg.matmul()` function

`tf.linalg.matmul` | TensorFlow Core v2.8.0

Multiplies matrix a by matrix b, producing a * b.

 https://www.tensorflow.org/api_docs/python/tf/linalg/matmul

```
print(tensor)
tf.matmul(tensor, tensor) # This is done by doing dot operation

tensor * tensor # different result since it is element wise operation

# Matrix multiplication with Python operator "@"
tensor @ tensor
```


Matrix Multiplication with tensors of different shapes

```
# Create a tensor (3, 2)
X = tf.constant([[1,2],
                 [3,4],
                 [5,6]])
# Create another tensor (3, 2)
Y = tf.constant([[7,8],
                 [9,10],
                 [11,12]])

X, Y

# Try to matrix multiply tensors of same shape
X @ Y # Error the inner dimensions must match
```

We can fix this by changing the shape of Y

tf.reshape | TensorFlow Core v2.8.0
Reshapes a tensor.
 https://www.tensorflow.org/api_docs/python/tf/reshape

```
# Change the shape of Y
tf.reshape(Y, shape=(2, 3))

# See the shape of original Y and reshaped Y
X.shape, tf.reshape(Y, shape=(2, 3)).shape
```

```
# Lets try to multiply X by reshaped Y
X @ tf.reshape(Y, shape=(2,3))

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

If we change the shape of X, the result will be different.

```
tf.matmul(tf.reshape(X,shape=(2,3)), Y) # The result is different - shape (2, 3)
tf.reshape(X, shape=(2,3)).shape, Y.shape
```




Note: When we are dealing with matrix multiplications using dot product, it really depends on which tensor you manipulate.

Tensor transformations - transpose

tf.transpose | TensorFlow Core v2.8.0

Transposes a, where a is a Tensor.

 https://www.tensorflow.org/api_docs/python/tf/transpose

We can do the same with `tf.transpose`. However, it is slightly different to what the reshape is.

```
X, tf.transpose(X), tf.reshape(X, shape=(2,3))
```

The difference between transpose and reshape is that transpose flips the axis, whereas reshape shuffles tensors around into the shape that we want.

The dot product

Matrix multiplication is also referred to as the dot product.

We can perform matrix multiplication using:

- `tf.matmul()` - https://www.tensorflow.org/api_docs/python/tf/linalg/matmul
- `tf.tensordot()` - https://www.tensorflow.org/api_docs/python/tf/tensordot

```
# Perform the dot product on X and Y (requires X or Y to be transposed)
tf.tensordot(tf.transpose(X), Y, axes=1)

# Perform matrix multiplication between X and Y (transposed)
tf.matmul(X, tf.transpose(Y))

# Perform matrix multiplication between X and Y (reshaped)
tf.matmul(X, tf.reshape(Y, shape=(2,3)))

# Check the values of Y, reshape ^ and transposed Y
print("Normal Y")
print(Y, "\n") # \n is for newline

print("Y reshaped to (2,3)")
print(tf.reshape(Y, shape=(2,3)), "\n")

print("Y transposed")
print(tf.transpose(Y), "\n")
```

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




Note: Generally, when performing matrix multiplication on two tensors and one of the axes doesn't line up, you will transpose (rather than reshape) one of the tensors to satisfy the matrix multiplication rules.

Changing the datatypes of tensors

TensorFlow mixed precisions

Mixed precision | TensorFlow Core

Mixed precision is the use of both 16-bit and 32-bit floating-point types in a model during training to make it run faster and use less memory. By keeping certain parts of the model in the 32-bit types for numeric stability, the model will have a lower step time and train

 https://www.tensorflow.org/guide/mixed_precision



- In TensorFlow `int32` datatype is a default
- With a modern piece of hardware, we can do `float16`, which takes 16 bits of memory instead. Modern accelerators can run operations faster in the 16-bit dtypes.

```
# Create a new tensor with default datatype (float32)
B = tf.constant([1.7, 7.4])
B.dtype

C = tf.constant([7, 10])
C, C.dtype
```

To change a datatype of a tensor, we use `tf.cast`.

tf.cast | TensorFlow Core v2.8.0

Casts a tensor to a new type.

 https://www.tensorflow.org/api_docs/python/tf/cast

```
# Change from float32 to float16 (reduced precision)
D = tf.cast(B, dtype=tf.float16)
D, D.dtype

# Change from int32 to float32
E = tf.cast(C, dtype=tf.float32)
```

```
E
```

```
E_float16 = tf.cast(E, dtype=tf.float16)  
E_float16
```



If we had a tensor of a million elements and we reduced the floating point size from 32 to 16. We would basically saved a half the amount of space our tensor is taking up on memory, allowing a hardware accelerator to make calculations on it potentially, twice fast

Aggregating tensors

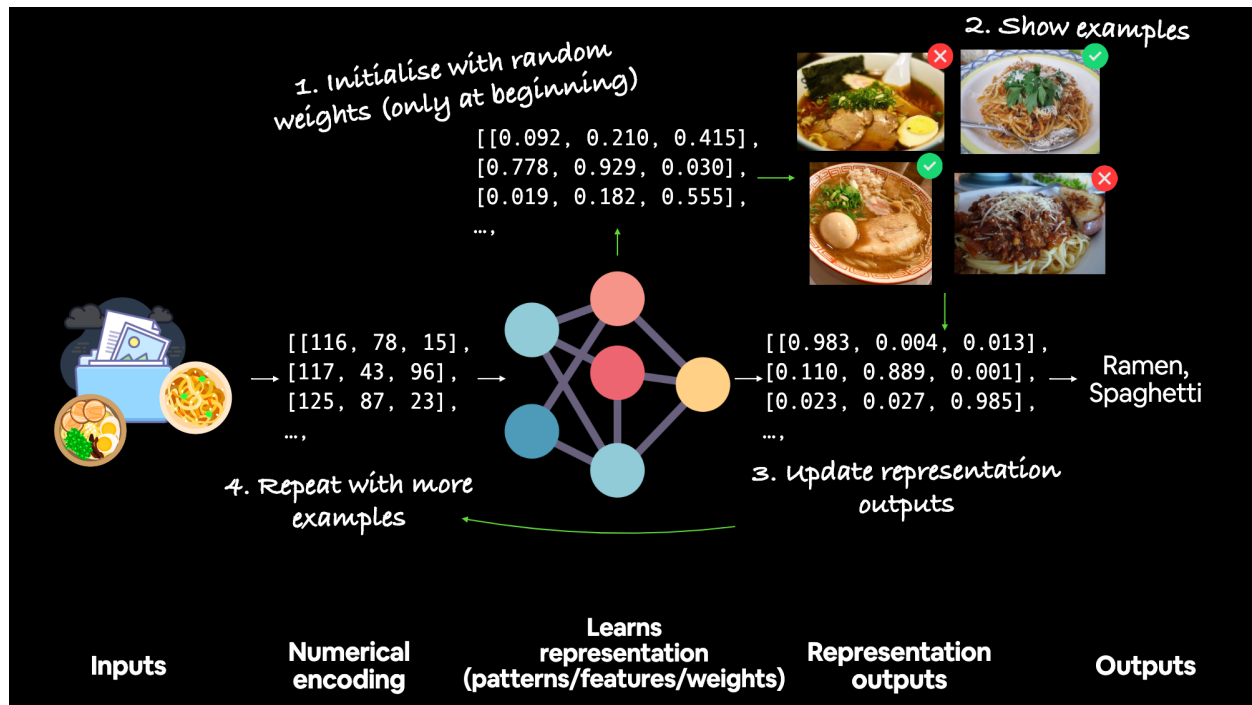
Aggregating tensors = condensing them from multiple values down to a smaller amount of values.

Forms of aggregations:

- Maximum
- Minimum
- Mean
- Sum
- Variance
- Standard Deviation

```
# Create a random tensor with values between 0 and 100 of size 50  
E = tf.constant(np.random.randint(0,100, size=50))  
E  
  
tf.size(E), E.shape, E.ndim, E.dtype  
  
# Find the minimum  
tf.reduce_min(E)  
# Find the maximum  
tf.reduce_max(E)  
# Find the mean  
tf.reduce_mean(E)  
# Find the sum  
tf.reduce_sum(E)  
  
# Find the variance  
tf.math.reduce_variance(tf.cast(E, dtype=tf.float32))  
# Find the std (Standard Deviation)  
tf.math.reduce_std(tf.cast(E, dtype=tf.float32))
```


Find the position minimum and maximum



Why do we need to find position maximum and position minimum ? 🤔

Let's take a close look at step 3 "Updated representation outputs". There, oftentimes the representation outputs often referred as prediction probabilities.

Picture/Image	Spaghetti Label	Ramen Label	No Food
image1	0.983	0.004	0.013
image2	0.110	0.889	0.001
image3	0.023	0.027	0.985


The numbers correspond to probabilities of classifying each label.

```
# Create a new tensor for finding positional minimum and maximum
tf.random.set_seed(42)
F = tf.random.uniform(shape=[50])
F
```

`tf.argmax()`

tf.math.argmax | TensorFlow Core v2.8.0

Returns the index with the largest value across axes of a tensor.

 https://www.tensorflow.org/api_docs/python/tf/math/argmax

```
# Find the positional maximum - often referred as argmax - Numpy alternative - np.argmax()
tf.argmax(F)
# Index on our largest value position
F[tf.argmax(F)]
# Find the maximum value of F
tf.reduce_max(F)
# Check for equality
assert F[tf.argmax(F)] == tf.reduce_max(F)
```

`tf.argmin()`

`tf.math.argmax` | TensorFlow Core v2.8.0

Returns the index with the smallest value across axes of a tensor.

 https://www.tensorflow.org/api_docs/python/tf/math/argmin

```
# Find the positional minimum - often referred as argmin - Numpy alternative - np.argmin()
tf.argmin(F)
# Find the minimum using the positional minimum index
F[tf.argmin(F)]
```


Squeezing a tensor (removing all single dimensions)

```
# Create a tensor to get started
tf.random.set_seed(42)
G = tf.constant(tf.random.uniform(shape=[50]), shape=(1,1,1,1,50))
G, G.shape
```

```
G_squeezed = tf.squeeze(G)
G_squeezed, G_squeezed.shape
```

`tf.squeeze` | TensorFlow Core v2.8.0

Removes dimensions of size 1 from the shape of a tensor.

 https://www.tensorflow.org/api_docs/python/tf/squeeze


`tf.squeeze()` - removes dimensions of size 1 from the shape of a tensor.

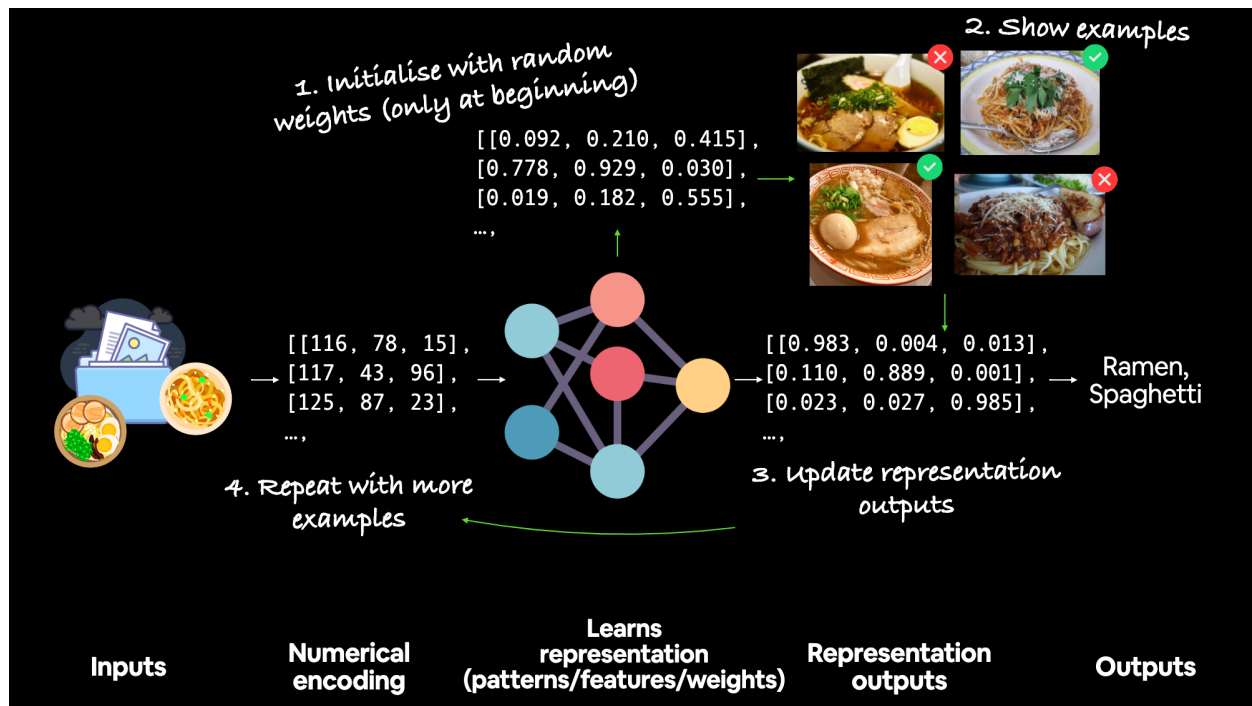
So, if our tensor has too many single dimensions and we want to reduce it (remove extra single dimensions) - we can use `tf.squeeze()` method.

One-hot Encoding tensors

Why One-Hot Encode Data in Machine Learning? - Machine Learning Mastery

Getting started in applied machine learning can be difficult, especially when working with real-world data. Often, machine learning tutorials will recommend or require that you prepare your data in specific ways before fitting a machine learning model. One good

 <https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>



When we are passing our data to neural networks, we have to find a way to numerically encode it.

One-hot encoding is a form of numerical encoding.

```
# Create a list of indices
some_list = [0,1,2,3] # could be red, green, blue, purple

# One hot encode our list of indices
tf.one_hot(some_list, depth=4)

# Specify custom values for one hot encoding
tf.one_hot(some_list, depth=4, on_value="I like Deep Learning", off_value="I also like Machine Learning.")
```

More Math Operations

- Squaring

- Log
- Square Root

```
# Create a new tensor
H = tf.range(1, 10)
H

# Square it
tf.square(H)

# Find the square root - https://www.tensorflow.org/api_docs/python/tf/math/sqrt
tf.sqrt(tf.cast(H, dtype=tf.float32)) # Method requires non-int dtype

# Find the log - https://www.tensorflow.org/api_docs/python/tf/math/log
tf.math.log(tf.cast(H, dtype=tf.float32)) # Method requires non-int dtype
```

Tensors and NumPy

TensorFlow interacts beautifully with NumPy arrays, and vice-versa

```
# Create a tensor directly from a NumPy array
J = tf.constant(np.array([3., 7., 10.]))
J

# Convert our tensor back to a NumPy array
np.array(J), type(np.array(J))

# Convert tensor J to a Numpy array
J.numpy(), type(J.numpy())

J = tf.constant([3.])
# Accessing an element - Numpy
J.numpy()[0]
```

The default types of each are slightly different

```
numpy_J = tf.constant(np.array([3., 7., 10.]))
tensor_J = tf.constant([3., 7., 10.])
# Check the datatypes of each
numpy_J.dtype, tensor_J.dtype
```