# Topic 1: Introduction to Tensorflow

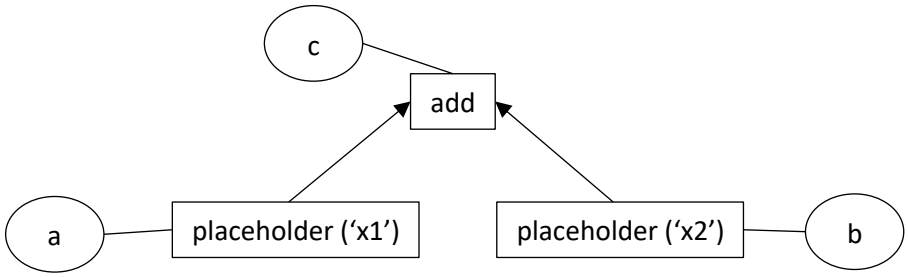
Tensorflow is a Python library for performing symbolic mathematical operations, which means that you create a graph structure that represents a number of connected mathematical operations that can be manipulated to do things like differentiation. It can also automatically evaluate the graph structure on the GPU for you, which makes it faster.

It is important to know what a tensor is: a structure of numbers organised as a grid with a certain number of dimensions. A 0-dimensional tensor is called a scalar, a 1-dimensional tensor is called a vector, a 2-dimensional tensor is called a matrix, a 3-dimensional tensor is called a 3tensor.

## 1. Graph and session

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/01_-_Graph_and_session.py

Tensorflow works using a graph and a session to evaluate the graph. Sessions take in inputs to the graph and return outputs. Here is the Tensorflow graph with Tensorflow nodes as rectangles and Python variables as ellipses.




Every time you use a node in an operation, a new node is created and added to the graph. You can visualise this graph in TensorBoard by running 'tensorboard --logdir=<path to folder with python file>' in the terminal and then opening the URL shown in the terminal (do this after running the program).

## 2. Tensor operations

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/02_-_Tensor_operations.py

There are several things you can do with a tensor in Tensorflow. Here are some useful operations:

- You can get the shape of a tensor using "tf.shape" (the same thing you set in the placeholder, but will be useful later).
- You can get a sub-tensor of a tensor by using indexing notation just like in NumPy. If "x" is a matrix then:
    - "x[0, 0]" gives you the first column of the first row (top left element)
    - "x[0, :]" gives you the whole first row

- "x[:, 0]" gives you the whole first column
- "x[:2, 1:3]" gives you the first two rows with the second and third columns

- You can concatenate two tensors together using "tf.concat". The tensors to concatenate must be given in a list and you also need to specify along which axis you want to concatenate them (one on top of the other or side by side?).
- You can reshape a tensor using "tf.reshape" so that it has the same number of elements but arranged into a different shape such as a 2 by 3 matrix being rearranged into a six element vectors.
- You can tile a tensor using "tf.tile" so that its rows or columns are replicated along their dimension (such as duplicating a matrix's rows under each other). You need to provide a list of how many time to replicate each axis such as "[2, 1]" meaning that you want to replicate the first dimension (the rows) twice but leave the second dimension (the columns) as is.

## 3. Tensor expressions

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/03_-_Tensor_expressions.py

You can compute expressions that work on tensors of any dimension instead of scalars. Note that the equation in the code is using an explicit conversion of Python constants into Tensorflow constants. We could instead let it happen implicitly (automatically) by writing '2*a + 1' but there would still be a conversion.

## 4. Getting/setting intermediate nodes

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/04_-_Getting-setting_intermediate_nodes.py

You can set a value for any intermediate node, as well as get the value of any intermediate node.

## 5. Variably sized tensors

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/05_-_Variably_sized_tensors.py

Tensors do not need to have a fixed size. Setting a size to 'None' will allow you to use any number of elements in that dimension. Any other expressions that depend on that tensor will change their size accordingly.

## 6. Variables

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/06_-_Variables.py

It is usually the case in machine learning that we need to use a model with parameters that are adjusted and learned so that we get a desired behaviour, that is, how inputs are mapped into outputs. In Tensorflow these parameters are called variables. Variables are not passed in with the inputs but are stored within the session. They are not the variables in the sense of Python's variables but are special nodes in the graph.

There are also special nodes for setting variables and the values set for the variables are stored in the session.

## 7. Initialisers

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/07_-_Initialisers.py

Multiple variables can be initialised together using an initialiser node.

## 8. Variable names and scopes

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/08_-_Variable_names_and_scopes.py

Variables must have unique names in order to be able to refer to them. In order to avoid coming up with long names we can organize them into variable scopes which act like folders. A variable in one scope can have the same name as another variable in another scope. It is possible to get a variable by its name by using 'get_tensor_by_name'. Every variable name ends with a ':0' such as 'varname:0'. If the variable is in a scope then its name will be 'scopename/varname:0'. If the scope is nested within another scope then its name will be 'scopename1/scopename2/varname:0'.

## 9. Saving and restoring variables

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/09_-_Saving_and_restoring_variables.py

You can save and load variables for later use by using a 'saver' graph node. Remember that variable values are session specific, so closing a session will destroy the variable values. If you want to use the same variable values in a different session you will need to either manually set them or load them from a saved model. The saver node is useful for publishing or loading a trained model or even for saving your half-trained model's variables in order to resume training if the program crashes or if the power goes out.

## 10. Polynomials

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/10_-_Polynomials.py

Here is an example of a model consisting of a polynomial.

## 11. Plotting with matplotlib

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/11_-_Plotting_with_matplotlib.py

Plotting a polynomial with matplotlib.

## 12. Gradients

https://github.com/mtanti/deeplearningtutorial/blob/master/tensorflow_v1/01_-_Introduction_to_Tensorflow/12_-_Gradients.py

Gradients are very useful in machine learning as they help you find the lowest point in a function or even the most sensitive input in a function (a high gradient means that a small change in the input will give a big change in the output). Tensorflow has functions for automatically finding the gradient of a graph. Unfortunately, we can only find the gradient of a scalar. It can be with respect to any shape tensor but the node we are finding the gradient of must be a single number, otherwise the gradient function will find the gradient of the sum of all elements in the tensor (it does not produce a Jaccard matrix).