< Deep Learning - PART1 TF2 Basics >

Ch 2. Workshop - Intro to TensorFlow 2.0 Programming

2021/10/01

[Reference]

1. TensorFlow Core - Tutorials : **`TensorFlow 2 quickstart for experts`** , 2019. https://www.tensorflow.org/tutorials/quickstart/advanced (https://www.tensorflow.org/tutorials/quickstart/advanced)
2. TensorFlow Core - Guides : **`Better performance with tf.function and AutoGraph`** https://www.tensorflow.org/guide/function (https://www.tensorflow.org/guide/function)
3. 程式前沿，**Python装飾器的函數語言程式設計詳解** https://codertw.com/%e7%a8%8b%e5%bc%8f%e8%aa%9e%e8%a8%80/372927/ (https://codertw.com/%e7%a8%8b%e5%bc%8f%e8%aa%9e%e8%a8%80/372927/)
4. **`TensorFlow2`基础操作篇：创建`Tensor`** https://zhuanlan.zhihu.com/p/68223806 (https://zhuanlan.zhihu.com/p/68223806)
5. **`tf.function`** 和 **`Autograph`使用指南`-Part 3`** https://zhuanlan.zhihu.com/p/68279357 (https://zhuanlan.zhihu.com/p/68279357)

< Content >

---

1. Running `TensorFlow 2 quickstart for experts`

- `advanced.ipynb` on Colab : https://www.tensorflow.org/tutorials/quickstart/advanced (https://www.tensorflow.org/tutorials/quickstart/advanced)

> Additional tutorials:

- **`tf.Tensor` -**
  - A Tensor is a multi-dimensional array.
  - Similar to NumPy ndarray objects, `tf.Tensor` objects have a data type and a shape.
  - Additionally, `tf.Tensor` s can reside in accelerator memory (like a GPU).
  - TensorFlow offers a rich library of operations (`tf.add` , `tf.matmul` , `tf.linalg.inv` etc.) that consume and produce `tf.Tensor` s. These operations automatically convert native Python types.
- `Customization basics: tensors and operations` ** **- `tf.Tensor`**
  https://www.tensorflow.org/tutorials/customization/basics
  (https://www.tensorflow.org/tutorials/customization/basics)

- `Distributed training with Keras`
  https://www.tensorflow.org/tutorials/distribute/keras
  (https://www.tensorflow.org/tutorials/distribute/keras)

---

2. TensorFlow 2.0 Programming Related

2.1 `tf.data.Dataset`

https://www.tensorflow.org/api_docs/python/tf/data/Dataset
(https://www.tensorflow.org/api_docs/python/tf/data/Dataset)

`Class` **`Dataset`** - Represents a potentially large set of elements.

Use `tf.data` to batch and shuffle the dataset:

For example :

```
train_ds = tf.data.Dataset.from_tensor_slices((x_train, y_train))
                          .shuffle(10000).batch(32)
```

[NOTE]:

- `tf.data.Dataset.from_tensor_slices` - https://www.tensorflow.org/api_docs/python/tf/data/Dataset#from_tensor_slices (https://www.tensorflow.org/api_docs/python/tf/data/Dataset#from_tensor_slices)
- `shuffle()` - https://www.tensorflow.org/api_docs/python/tf/data/Dataset?version=stable#shuffle (https://www.tensorflow.org/api_docs/python/tf/data/Dataset?version=stable#shuffle)
- `batch()` - https://www.tensorflow.org/api_docs/python/tf/data/Dataset?version=stable#batch (https://www.tensorflow.org/api_docs/python/tf/data/Dataset?version=stable#batch)

2.2 `tf.keras`

- **`tf.keras`** is *TensorFlow's high-level API for building and training deep learning models*.

  - Module: `tf.keras` - https://www.tensorflow.org/api_docs/python/tf/keras (https://www.tensorflow.org/api_docs/python/tf/keras)
  - Subclass: `Keras` - https://www.tensorflow.org/guide/keras#model_subclassing (https://www.tensorflow.org/guide/keras#model_subclassing)

- **3 key advantages**:

  - *User-friendly*
  - *Modular and composable*
  - *Easy to extend*

[Starter Tutorials for `tf.keras`] :

- `Basic classification: Classify images of clothing`
  https://www.tensorflow.org/tutorials/keras/classification
  (https://www.tensorflow.org/tutorials/keras/classification)
- `Text classification with TensorFlow Hub: Movie reviews`
  https://www.tensorflow.org/tutorials/keras/text_classification_with_hub
  (https://www.tensorflow.org/tutorials/keras/text_classification_with_hub)
- `Text classification with preprocessed text: Movie reviews`
  https://www.tensorflow.org/tutorials/keras/text_classification
  (https://www.tensorflow.org/tutorials/keras/text_classification)
- `Basic regression: Predict fuel efficiency`
  https://www.tensorflow.org/tutorials/keras/regression
  (https://www.tensorflow.org/tutorials/keras/regression)
- `Explore overfit and underfit`
  https://www.tensorflow.org/tutorials/keras/overfit_and_underfit
  (https://www.tensorflow.org/tutorials/keras/overfit_and_underfit)
- `Save and load models`
  https://www.tensorflow.org/tutorials/keras/save_and_load
  (https://www.tensorflow.org/tutorials/keras/save_and_load)

- **`tf.keras.losses.SparseCategoricalCrossentropy()`** :
  https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy
  (https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy)

  - **Class `SparseCategoricalCrossentropy` -** Computes the crossentropy loss between the labels and predictions.
  - **Use this crossentropy loss function when there are two or more label classes.**
  - **We expect labels to be provided as integers.**

  - **If you want to provide labels using `one-hot` representation, please use `CategoricalCrossentropy` loss.**

  - There should be `# classes` floating point values per feature for `y_pred` and a single floating point value per feature for `y_true` .

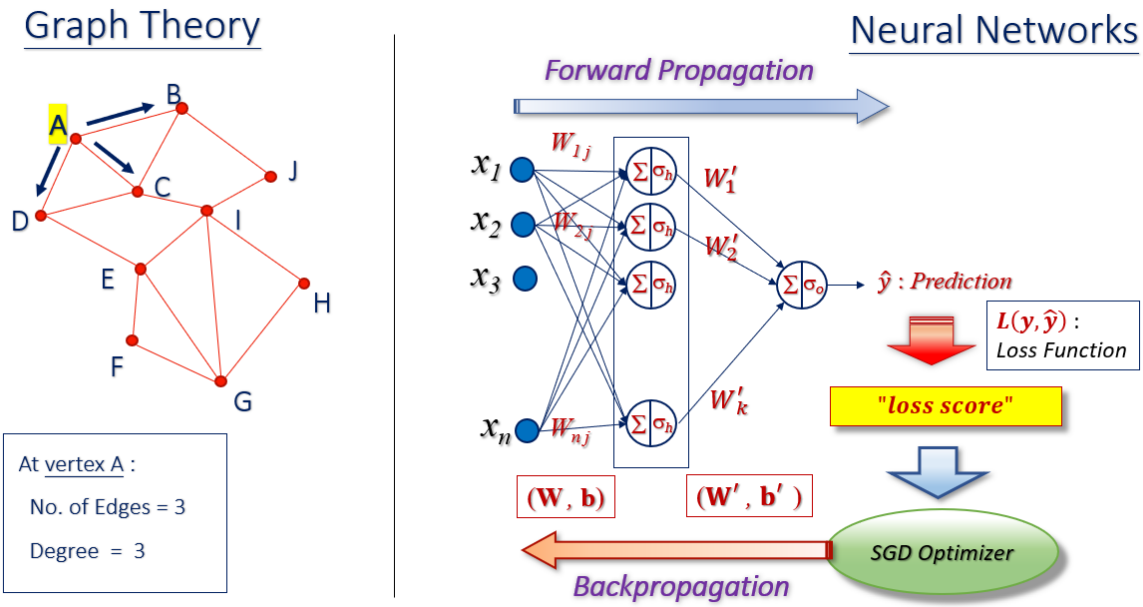
- Module: **`tf.keras.metrics`** https://www.tensorflow.org/api_docs/python/tf/keras/metrics
  (https://www.tensorflow.org/api_docs/python/tf/keras/metrics)

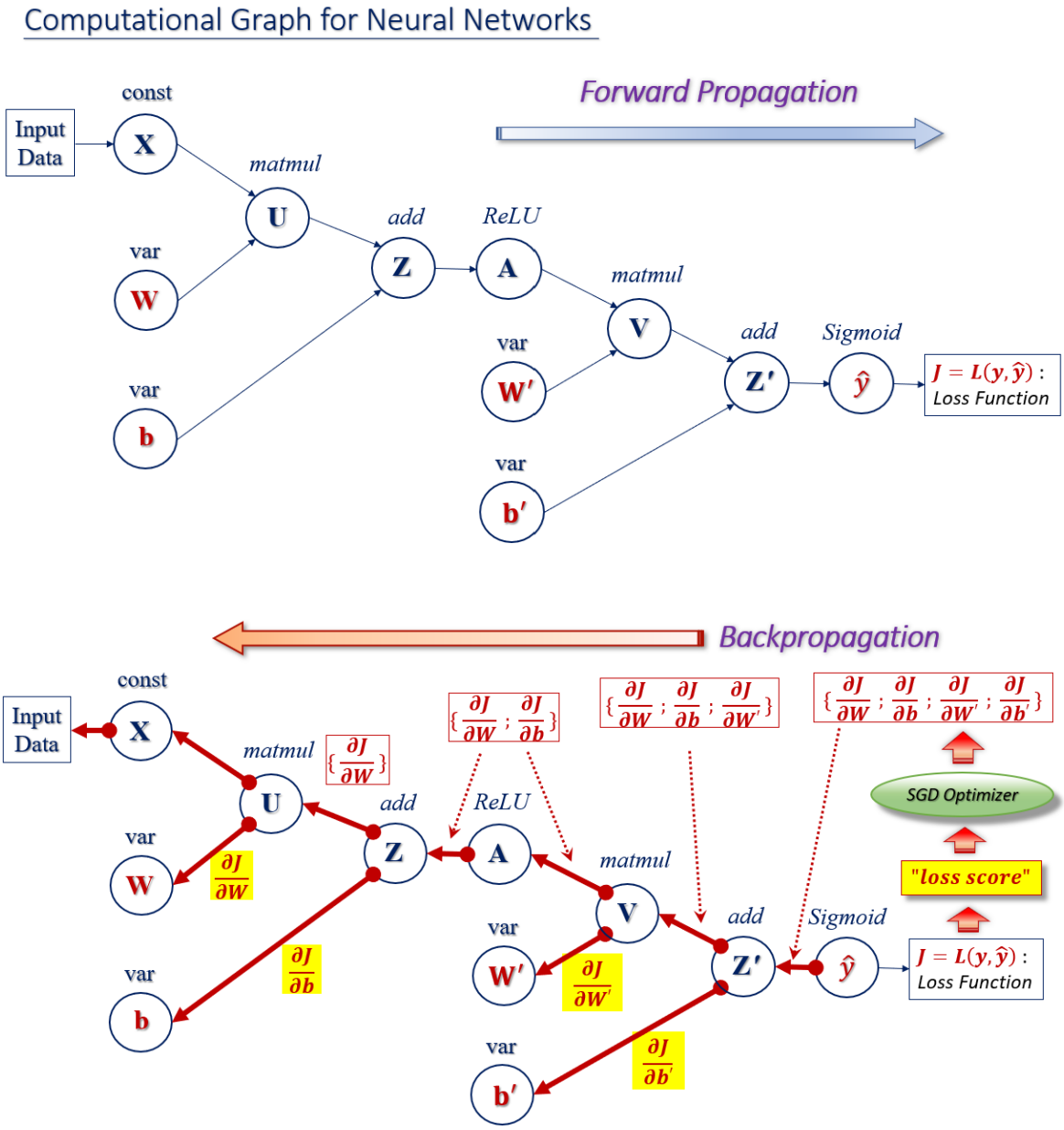
  Built-in metrics classes : (*to name a few*)

- **`class SparseCategoricalAccuracy`** : Calculates how often predictions matches integer labels. https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalAccuracy (https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalAccuracy)
- **`class CategoricalAccuracy`** : Calculates how often predictions matches labels.
- **`class Mean`** : Computes the (weighted) mean of the given values.

2.3 Computational Graph for Neural Networks

- **[Ref]: YouTube - `MLwPython_Lecture 01 - Perceptron, Logistic Regression & Neural Networks`** ,https://youtu.be/6AWs76fsKo4 (https://youtu.be/6AWs76fsKo4)
  - **[PPT & Code]:** https://drive.google.com/drive/folders/10AwHPBLsCFnWMCav5GCvoJ3eNqPoY (https://drive.google.com/drive/folders/10AwHPBLsCFnWMCav5GCvoJ3eNqPoY





- **`@tf.function`** Decorator ：一種 Python 函數語言程式設計的技巧。

> **[REFERENCE]** :
>
> 2.1 TensorFlow Core - Guides : **`Better performance with tf.function and AutoGraph`** https://www.tensorflow.org/guide/function (https://www.tensorflow.org/guide/function)
>
> 2.2 程式前沿，**Python裝飾器的函數語言程式設計詳解** https://codertw.com/%e7%a8%8b%e5%bc%8f%e8%aa%9e%e8%a8%80/372927/ (https://codertw.com/%e7%a8%8b%e5%bc%8f%e8%aa%9e%e8%a8%80/372927/)

Decorator 的本質 : (from **Ref. 2.2**)

當你用一個 @decorator 來修飾某個函式 func 時，如下所示:

```
@decorator
def func():
    pass
```

其直譯器會解析成下面這樣的語句：

```
func = decorator(func)
```

In [1]:

```python
# Example from 程式前沿, "Python裝飾器的函數語言程式設計詳解"
# https://codertw.com/%e7%a8%8b%e5%bc%8f%e8%aa%9e%e8%a8%80/372927/

def hello(fn):
    def wrapper():
        print("hello, %s" % fn.__name__)
        fn()
        print("goodby, %s" % fn.__name__)
    return wrapper

print('@hello: foo() \n')

@hello        # foo = hello(foo)
def foo():
    print("i am foo")

print('Calling foo()... \n')
foo()
```

```
@hello: foo()

Calling foo()...

hello, foo
i am foo
goodby, foo
```

AutoGraph

- AutoGraph 是 TF 提供的一個非常具有前景的工具,它能夠將一部分 python 語法的代碼轉譯成高效的圖表示代碼.
- 由於從 TF 2.0 開始, TF 將會預設使用動態圖(eager execution),因此利用 AutoGraph , 在理想情況下 ,能讓我們方便、靈活使用動態圖撰寫程式,同時利用靜態圖方式高效、穩定執行程式。

[REFERENCE] :

- TensorFlow Core - Guides : **`Better performance with tf.function and AutoGraph`** https://www.tensorflow.org/guide/function (https://www.tensorflow.org/guide/function))
- NLPer, 機器學習谷歌開發者專家, **`tf.function和Autograph使用指南-Part 1`** https://zhuanlan.zhihu.com/p/67192636 (https://zhuanlan.zhihu.com/p/67192636)
- Github, **TensorFlow 2.0: Functions, not Sessions.** https://github.com/tensorflow/community/blob/master/rfcs/20180918-functions-not-sessions-20.md (https://github.com/tensorflow/community/blob/master/rfcs/20180918-functions-not-sessions-20.md)
- TensorFlow - Guides : **`Keras and AutoGraph`** https://www.tensorflow.org/guide/function#keras_and_autograph (https://www.tensorflow.org/guide/function#keras_and_autograph)

- `TF2.0` 的**其中一個重要改變就是去除 `TF 1.x` 版本的 `tf.Session`** .

- 這個改變會使得開發者採用更好的方式來撰寫程式 : 不必再用 `tf.Session` 來執行 `TF` 程式，只需要在一個 python 函數，加上一個簡單的 裝飾器 (decorator) 即可。

- 在 `TF2.0` 裡面，**如果需要構建計算圖，我們只需要給 python 函數加上 `@tf.function` 的裝飾器**.

- **這個自動將python代碼轉成圖表示代碼的工具就叫做AutoGraph**.

- 在 `TF2.0` 中，如果一個函數被 **`@tf.function`** 裝飾了，那麼 **`AutoGraph`** 將會被自動調用，從而將 python 函數轉換成可執行的圖表示.

3. More on `tf.function` & AutoGraph

TensorFlow Core - Guide : "Better performance with tf.function and AutoGraph"

https://www.tensorflow.org/guide/function#keras_and_autograph (https://www.tensorflow.org/guide/function#keras_and_autograph)

Eager Execution

In [2]:

```
import tensorflow as tf
import numpy as np
```

In [3]:

```
print(tf.__version__)

# Checking Eager Execution mode...
print('Q: Is Eager Execution active? A: {}'.format(tf.executing_eagerly()))
```

```
2.0.0
Q: Is Eager Execution active? A: True
```

In [4]:

```
# Building a computation graph...
a = tf.constant(2)
b = tf.constant(3)
c = tf.constant(5)

d = tf.multiply(a, b)
e = tf.add(b, c)
f = tf.subtract(d, e)

# Launching the computation graph dynamically with Eager Execution.
print(a)
print('a = {}'.format(a))
print(f)
print('f = {}'.format(f))
```

```
tf.Tensor(2, shape=(), dtype=int32)
a = 2
tf.Tensor(-2, shape=(), dtype=int32)
f = -2
```

Using `tf.function` decorator

> - When you annotate a function with **`tf.function`** , you can still call it like any other function.

- **But `tf.function` will be compiled into a graph**, which means you get the benefits of faster execution, running on GPU or TPU, or exporting to SavedModel.

In [5]:

```
@tf.function
def Logit_Regress(w, x):
  return tf.nn.sigmoid(tf.matmul(w, x)+b)

w = tf.random.uniform((1, 3))
x = tf.random.uniform((3, 1))
b = tf.random.uniform((1,), minval=2, maxval=3)

print('x : ', x)
print('w : ', w)
print('b = ', b)
print('\n-----------')

print('Logit_Regress(w, x) = {} \n'.format(Logit_Regress(w, x)))
print(Logit_Regress(w, x))
print('-----------')
Logit_Regress(w, x)
```

```
x :  tf.Tensor(
[[0.77874935]
 [0.67371106]
 [0.09093118]], shape=(3, 1), dtype=float32)
w :  tf.Tensor([[0.01682639 0.14035904 0.06817579]], shape=(1,
3), dtype=float32)
b =  tf.Tensor([2.1116662], shape=(1,), dtype=float32)

-----------
Logit_Regress(w, x) = [[0.90251887]]

tf.Tensor([[0.90251887]], shape=(1, 1), dtype=float32)
-----------
```

Out[5]:

```
<tf.Tensor: id=40, shape=(1, 1), dtype=float32, numpy=array([[0.
90251887]], dtype=float32)>
```

- **If we examine the result of the annotation, we can see that it's a special callable that handles all interactions with the TensorFlow runtime.**

In [6]:

```
Logit_Regress
```

Out[6]:

```
<tensorflow.python.eager.def_function.Function at 0x2017c094ec8>
```

- **If your code uses multiple functions, you don't need to annotate them all - any functions called from an annotated function will also run in graph mode.**

In [7]:

```
def linear_layer(x):
  return 2 * x + 1

@tf.function
def deep_net(x):
  return tf.nn.relu(linear_layer(x))

deep_net(tf.constant((1, 2, 3)))
```

Out[7]:

```
<tf.Tensor: id=52, shape=(3,), dtype=int32, numpy=array([3, 5,
7])>
```

- **Functions can be faster than eager code, for graphs with many small ops. But for graphs with a few expensive ops (like convolutions), you may not see much speedup.**

In [8]:

```
import timeit   # This module provides a simple way to time small bits of P
conv_layer = tf.keras.layers.Conv2D(100, 3)  # 2D Convolution Layers

@tf.function
def conv_fn(image):
    return conv_layer(image)

# 4D-Tensor:(samples, height, width, channels)
image = tf.zeros([1, 200, 200, 100])

# warm up
conv_layer(image); conv_fn(image)
print("Eager conv:", timeit.timeit(lambda: conv_layer(image), number=10))
print("Function conv:", timeit.timeit(lambda: conv_fn(image), number=10))
print("Note how there's not much difference in performance for convolutions
```

```
Eager conv: 1.1287187000000003
Function conv: 1.101934400000001
Note how there's not much difference in performance for convolut
ions
```

In [9]:

```
#----------------------------------------
# Long-Short-Term-Memory (LSTM) model - one of the RNNs
# LSTMCell - This class processes one step within the whole time sequence t
#            whereas tf.keras.layer.LSTM processes the whole sequence.
#----------------------------------------
lstm_cell = tf.keras.layers.LSTMCell(10)

@tf.function
def lstm_fn(input, state):
    return lstm_cell(input, state)

input = tf.ones([10, 10])
state = [tf.ones([10, 10])] * 2

# warm up
lstm_cell(input, state); lstm_fn(input, state)
print("eager lstm:", timeit.timeit(lambda: lstm_cell(input, state), number=
print("function lstm:", timeit.timeit(lambda: lstm_fn(input, state), numbe
```

```
eager lstm: 0.026034899999999084
function lstm: 0.004357699999999909
```

Using Python control flow

> - When using data-dependent control flow inside `tf.function` , you can use Python control flow statements and AutoGraph will convert them into appropriate TensorFlow ops. For example, `if` statements will be converted into `tf.cond()` if they depend on a `Tensor` .

- In the example below, `x` is a `Tensor` but the `if` statement works as expected:

In [10]:

```
@tf.function
def square_if_positive(x):
    if x > 0:
        x = x * x
    else:
        x = 0
    return x

print('square_if_positive(2) = {}'.format(square_if_positive(tf.constant(2)
print('square_if_positive(-2) = {}'.format(square_if_positive(tf.constant(-
```

```
square_if_positive(2) = 4
square_if_positive(-2) = 0
```

- **AutoGraph supports common Python statements like `while`, `for`, `if`, `break`, `continue` and `return`, with support for nesting.**

- That means you can use `Tensor` expressions in the condition of `while` and `if` statements, or iterate over a `Tensor` in a `for` loop.

In [11]:

```
@tf.function
def sum_even(items):
    s = 0
    for c in items:
        if c % 2 > 0:
            continue
        s += c
    return s

sum_even(tf.constant([10, 12, 15, 20]))
```

Out[11]:

```
<tf.Tensor: id=619, shape=(), dtype=int32, numpy=42>
```

- **AutoGraph also provides a low-level API for advanced users.**

For example we can use it to have a look at the generated code:

In [12]:

```
1 print(tf.autograph.to_code(sum_even.python_function))
```

```
def tf__sum_even(items):
  do_return = False
  retval_ = ag__.UndefinedReturnValue()
  with ag__.FunctionScope('sum_even', 'sum_even_scope', ag__.Con
versionOptions(recursive=True, user_requested=True, optional_fea
tures=(), internal_convert_user_code=True)) as sum_even_scope:
    s = 0

    def get_state_2():
      return ()

    def set_state_2(_):
      pass

    def loop_body(iterates, s):
      c = iterates
      continue_ = False

      def get_state():
        return ()

      def set_state(_):
        pass

      def if_true():
        continue_ = True
        return continue_

      def if_false():
        return continue_
      cond = c % 2 > 0
      continue_ = ag__.if_stmt(cond, if_true, if_false, get_stat
e, set_state, ('continue_',), ())

      def get_state_1():
        return ()

      def set_state_1(_):
        pass

      def if_true_1():
        s_1, = s,
        s_1 += c
        return s_1

      def if_false_1():
        return s
      cond_1 = ag__.not_(continue_)
      s = ag__.if_stmt(cond_1, if_true_1, if_false_1, get_state_
1, set_state_1, ('s',), ())
      return s,
    s, = ag__.for_stmt(items, None, loop_body, get_state_2, set_
state_2, (s,), ('s',), ())
```

```
    do_return = True
    retval_ = sum_even_scope.mark_return_value(s)
  do_return,
  return ag__.retval(retval_)
```

An example of more complicated control flow:

In [13]:

```
@tf.function
def fizzbuzz(n):
    for i in tf.range(n):
        if i % 3 == 0:
            tf.print('Fizz')
        elif i % 5 == 0:
            tf.print('Buzz')
        else:
            tf.print(i)

fizzbuzz(tf.constant(15))
```

```
Fizz
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
```

[Further Reading] : Keras and AutoGraph

- Ref : " `Keras and AutoGraph` "
  https://www.tensorflow.org/guide/function#keras_and_autograph
  (https://www.tensorflow.org/guide/function#keras_and_autograph)

In [14]:

```
class CustomModel(tf.keras.models.Model):

    @tf.function
    def call(self, input_data):
        if tf.reduce_mean(input_data) > 0:
            return input_data
        else:
            return input_data // 2


model = CustomModel()

model(tf.constant([-2, -4]))
```

Out[14]:

```
<tf.Tensor: id=727, shape=(2,), dtype=int32, numpy=array([-1, -
2])>
```