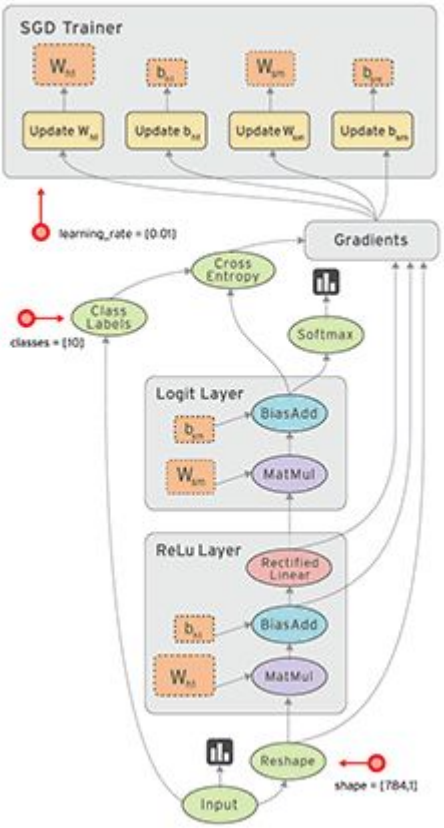
TensroFlow представляет собой библиотеку для создания и обучения моделей на основе машинного обучения.
Версии 1.x и 2.0 основаны на различных парадигмах реализации моделей. У каждого подхода есть свои преимущества и недостатки.

В основе подхода TensorFlow лежит направленный ацикличный граф (DAG), где узлом является операция, а направление -- это переход от выполненной операции к выполнению следующей.
Tensorflow 1.+ известен своим статическим графом, с которым очень тяжело работать в режиме debug, и, самое главное, не позволяет реализовывать динамическую архитектуру: если есть какие-то компоненты, которые могут возникнуть в процессе обучения, то вставить их в статический граф невозможно, так как вся архитектура и весь flow обучения предопределяется заранее. То есть сначала строиться вычислительный граф и затем по нему происходят вычисления необходимые в процессе обучения и валидации.
Но здесь есть одно большое преимущество, которое заставляет Community TensorFlow продолжать работать над версией 1.x -- это производительность такого подхода. Их документация и эксперименты указывает на то, что в production статический граф гораздо эффективнее, нежели динамический, а также статический граф позволяет легко и эффективно реализовывать распределенное обучение.

Graph
Рассмотрим что собой представляет граф:
С помощью класса Graph создается сама модель на языке TensorFlow:
1) реализуется ее архитектура;
2) создаются необходимые переменные, отвечающие за входные данные;
3) указывается порядок выполнения операций.



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

Tensorflow создает граф по умолчанию независимо от того, было ли явно указано создание графа или нет.
Tensorflow отслеживает, что создано под каким контекстом графа.

В данном примере созданный граф переопределяет граф по умолчанию, который считается переопределенным внутри объявленной области видимости (или внутри объявленного контекста).

```
graph = tf.Graph()
with graph.as_default():
    tmp = tf.constant(5.)

assert tmp.graph is graph
```

После того как контекст явно созданного графа закончился (все то, что не входит в область видимости *with graph.as_default()*), то новые операции будут добавляться в граф по умолчанию. То есть созданный граф становится на время по умолчанию для того, чтобы при многопоточности не возникло конфликтов и было явно видно, к какому графу принадлежат созданные операции. Но глобальный граф по умолчанию всегда существует в программе.

Если явно не указывать создание графа, Tensorflow по умолчанию создают граф, и в него помещает все созданные переменные, операции.

```
tmp = tf.constant(5.)
assert tmp.graph is tf.get_default_graph()
```

Session

Для того чтобы запустить вычисления объявленного графа, необходимо создать сессию (Session), которая запускает все вычислительные операции.
В данном примере сессия использует граф по умолчанию, так как явно граф не был создан.

```
tmp = tf.multiply(2, 3)

session = tf.Session()
output = session.run(tmp)

print(output) # 6
```

Для того чтобы выполнялись операции, указанные в определенном графе, необходимо указать сессии, с каким графом работать.

```
graph = tf.Graph()
with graph.as_default():
    tmp = tf.multiply(2, 3)

session = tf.Session(graph=graph)
output = session.run(tmp)

print(output) # 6
```

Также при создании сессии можно указать, на каком девайсе запускать вычисления: CPU или GPU.

GPU config

```
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=1.0)
session = tf.Session(graph=graph, config=tf.ConfigProto(gpu_options=gpu_options))
```

CPU config

```
session_conf = tf.ConfigProto(
    device_count={'CPU': 1, 'GPU': 0},
    allow_soft_placement=True,
    log_device_placement=False
)
session = tf.Session(graph=graph, config=session_conf)
```

При запуске сессии, Tensorflow анализирует граф целиком, но вычисляет только те операции, которые указаны были при запуске сессии (поданы в метод run).

Placeholder

Placeholder -- переменная, которая предназначены для входных данных. При создании графа данные переменные участвуют для передачи данных, которые инициализируются перед вычислением графа.

Для того чтобы нужные значения попали к соответствующим placeholder, при запуске вычисления графа необходимо подать в сессию feed_dict:

```
tmp = tf.placeholder(dtype=tf.int32, shape=(None), name='temp_variable')
out = tf.matmul(tmp, tmp)

session = tf.Session()
output = session.run(out, feed_dict={tmp: np.ones((2, 2))})

print(output)  #[[2 2] [2 2]]
```

Если в feed_dict не указать значения для placeholder, то код упадет с ошибкой. Каждый созданный placeholder должен быть проинициализирован при запуске вычислительного графа.

Variable
Объект класса Variable позволяет создавать переменные, которые могут быть как и обучаемыми, так и не обучаемыми, участвующие в вычислительных процессах (графе)

```
temp = tf.Variable(initial_value=5., dtype=tf.float32,
                   trainable=False, name='temp')

output = tf.multiply(temp, temp)
```

Tensor (operations)
Class Tensor, тип данных Tensorflow, где каждый выход операции или объявлении переменной является экземпляром класса Tensor.

Сделать картинку графа, где Node -- это Operation

Control Dependencies

Позволяет задать порядок выполнения операций:
На вход *control_dependencies* можно подать как сами операция, так и тензоры. Control dependencies возвращает контекст, внутри которого можно объявить те операции и тензоры, которые будут выполняться в вычислительном графе после выполнения операций, которые пришли на вход.
Можно сделать вложенные *control_dependencies* и тогда, выстраивается иерархия выполнения операций.

Можно стереть все зависимости, подав на вход None.

Важно, чтобы операции создавались строго внутри контекста *control_dependencies*, если необходимо создать зависимость от операций, созданных ранее. Аналог того, что если хотим, чтобы операции принадлежали графу, который был создан явно, а не графу по умолчанию, то операции создаются внутри контекста данного графа, иначе будут относиться к графу по умолчанию.

Eager Execution

Среда для выполнения операция без построения графа в режиме реального времени. Поток управления операциями происходит с помощью Python, а не графом.

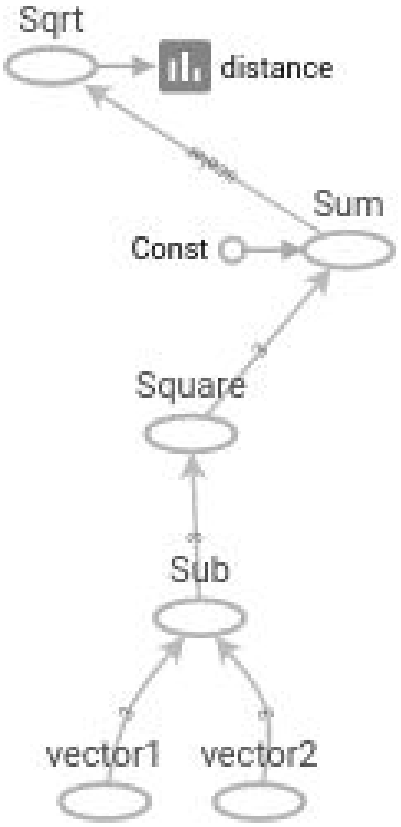
`tf.enable_eager_execution()  <=` Запускает EagerExecution Environment

Особенности:

- Use Python control flow instead of Graph control flow
- Works with Python objects and NumPy arrays (convert to tf.Tensor)
- tf.Tensor refers concrete value
- tf.Tensor.numpy() returns NumPy array
- Easy to control backpropagation
- *Program* state is *not* stored in *global collection* and managed by *tf.Session* anymore!

Важно отметить, что при использовании *placeholder* в *Eager Execution* нет никакого смысла, так как операции вычислить в режиме реального времени невозможно, поскольку данные не были еще поданы. Будут вычисляться только те переменные, которые не зависят от placeholders.

Пример графа для программы для вычисления евклидового расстояния:



```
def run(data):
    graph = tf.Graph()
    with graph.as_default():
        v1_ph = tf.placeholder(shape=[DIM], dtype=tf.float32, name='vector1')
        v2_ph = tf.placeholder(shape=[DIM], dtype=tf.float32, name='vector2')

        sub = tf.subtract(v1_ph, v2_ph)
        square = tf.square(sub)
        sum = tf.reduce_sum(square)

        distance = tf.sqrt(sum)

        tf.summary.scalar('distance', distance)

        summary_merged_op = tf.summary.merge_all()
        tensorboard_writer = tf.summary.FileWriter('./logs', tf.get_default_graph())

    if USE_GPU:
        gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=GPU_MEMORY_FRACTION)
        session_conf = tf.ConfigProto(allow_soft_placement=True,
                                      log_device_placement=True,
                                      gpu_options=gpu_options)
    else:
        session_conf = tf.ConfigProto(
            device_count={'CPU': 1, 'GPU': 0},
            allow_soft_placement=True,
            log_device_placement=True
        )
    session = tf.Session(graph=graph, config=session_conf)

    with session.as_default():
        summary_, distance_output = session.run([summary_merged_op, distance],
                                      feed_dict={v1_ph: data['v1'],
                                                 v2_ph: data['v2']})
        tensorboard_writer.add_summary(summary_, 0)
        print(distance_output)

    return
```