



File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3

Open in Colab

Autograph: Graphs for complex code

In this ungraded lab, you'll go through some of the scenarios from the lesson [Creating graphs for complex code](#).

Imports

```
In [1]:  try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
except Exception:
    pass
import tensorflow as tf
```

As you saw in the lectures, seemingly simple functions can sometimes be difficult to write in graph mode. Fortunately, Autograph generates this complex graph code for us.

- Here is a function that does some multiplication and addition.

```
In [2]:  a = tf.Variable(1.0)
b = tf.Variable(2.0)

@tf.function
def f(x,y):
    a.assign(y * b)
    b.assign_add(x * a)
    return a + b

print(f(1.0, 2.0))

print(tf.autograph.to_code(f.python_function))

tf.Tensor(10.0, shape=(), dtype=float32)
def tf_f(x, y):
    with ag__.FunctionScope('f', 'fscope', ag__.ConversionOptions(recursive=True, user_requested=True, optional_features=(), internal_convert_user_code=True)) as fscope:
        do_return = False
        retval_ = ag__.UndefinedReturnValue()
        ag__.converted_call(ag__.ld(a).assign, ((ag__.ld(y) * ag__.ld(b))), None, fscope)
        ag__.converted_call(ag__.ld(b).assign_add, ((ag__.ld(x) * ag__.ld(a))), None, fscope)
    try:
        do_return = True
        retval_ = (ag__.ld(a) + ag__.ld(b))
    except:
        do_return = False
        raise
    return fscope.ret(retval_, do_return)
```

- Here is a function that checks if the sign of a number is positive or not.

```
In [3]:  @tf.function
def sign(x):
    if x > 0:
        return 'Positive'
    else:
        return 'Negative or zero'

print("Sign = {}".format(sign(tf.constant(2))))
print("Sign = {}".format(sign(tf.constant(-2))))

print(tf.autograph.to_code(sign.python_function))

Sign = b'Positive'
Sign = b'Negative or zero'
def tf_sign(x):
    with ag__.FunctionScope('sign', 'fscope', ag__.ConversionOptions(recursive=True, user_requested=True, optional_features=(), internal_convert_user_code=True)) as fscope:
        do_return = False
        retval_ = ag__.UndefinedReturnValue()

    def get_state():
        return (retval_, do_return)

    def set_state(vars_):
        nonlocal retval_, do_return
        (retval_, do_return) = vars_

    def if_body():
        nonlocal retval_, do_return
        try:
            do_return = True
            retval_ = 'Positive'
        except:
            do_return = False
            raise

    def else_body():
        nonlocal retval_, do_return
        try:
            do_return = True
            retval_ = 'Negative or zero'
        except:
            do_return = False
            raise
    ag__.if_stmt((ag__.ld(x) > 0), if_body, else_body, get_state, set_state, ('retval_', 'do_return'), 2)
return fscope.ret(retval_, do_return)
```

- Here is another function that includes a while loop.

```
In [4]: @tf.function
def f(x):
    while tf.reduce_sum(x) > 1:
        tf.print(x)
        x = tf.tanh(x)
    return x

print(tf.autograph.to_code(f.python_function))

def tf_f(x):
    with ag_.FunctionScope('f', 'fscope', ag_.ConversionOptions(recursive=True, user_requested=True, optional_features=()), internal_convert_user_code=True) as fscope:
        do_return = False
        retval_ = ag_.UndefinedReturnValue()

    def get_state():
        return (x,)

    def set_state(vars_):
        nonlocal x
        (x,) = vars_

    def loop_body():
        nonlocal x
        ag_.converted_call(ag_.ld(tf).print, (ag_.ld(x),), None, fscope)
        x = ag_.converted_call(ag_.ld(tf).tanh, (ag_.ld(x),), None, fscope)

    def loop_test():
        return (ag_.converted_call(ag_.ld(tf).reduce_sum, (ag_.ld(x),), None, fscope) > 1)
    ag_.while_stmt(loop_test, loop_body, get_state, set_state, ('x'), {})

    try:
        do_return = True
        retval_ = ag_.ld(x)
    except:
        do_return = False
        raise
    return fscope.ret(retval_, do_return)
```

- Here is a function that uses a for loop and an if statement.

```
In [5]: @tf.function
def sum_even(items):
    s = 0
    for c in items:
        if c % 2 > 0:
            continue
        s += c
    return s

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

def tf_sum_even(items):
    with ag_.FunctionScope('sum_even', 'fscope', ag_.ConversionOptions(recursive=True, user_requested=True, optional_features=()), internal_convert_user_code=True) as fscope:
        do_return = False
        retval_ = ag_.UndefinedReturnValue()
        s = 0

    def get_state_2():
        return (s,)

    def set_state_2(vars_):
        nonlocal s
        (s,) = vars_

    def loop_body(itr):
        nonlocal s
        c = itr
        continue_ = False

        def get_state():
            return (continue_,)

        def set_state(vars_):
            nonlocal continue_
            (continue_,) = vars_

        def if_body():
            nonlocal continue_
            continue_ = True

        def else_body():
            nonlocal continue_
            pass
        ag_.if_stmt(((ag_.ld(c) % 2) > 0), if_body, else_body, get_state, set_state, ('continue_'), 1)

        def get_state_1():
            return (s,)

        def set_state_1(vars_):
            nonlocal s
            (s,) = vars_

        def if_body_1():
            nonlocal s
            s = ag_.ld(s)
            s += c

            def else_body_1():
                nonlocal s
                pass
            ag_.if_stmt(ag_.not_(continue_), if_body_1, else_body_1, get_state_1, set_state_1, ('s'), 1)
            continue_ = ag_.Undefined('continue_')
            c = ag_.Undefined('c')
            ag_.for_stmt(ag_.ld(items), None, loop_body, get_state_2, set_state_2, ('s'), {'iterate_names': 'c'})

        try:
            do_return = True
            retval_ = ag_.ld(s)
        except:
            do_return = False
            raise
    return fscope.ret(retval_, do_return)
```

Print statements

Tracing also behaves differently in graph mode. First, here is a function (not decorated with `@tf.function` yet) that prints the value of the input parameter.

`f(2)` is called in a for loop 5 times, and then `f(3)` is called.

```
In [6]: ┌─ def f(x):
    print("Traced with", x)

    for i in range(5):
        f(2)

    f(3)
    Traced with 2
    Traced with 3
```

If you were to decorate this function with `@tf.function` and run it, notice that the print statement only appears once for `f(2)` even though it is called in a loop.

```
In [7]: ┌─ @tf.function
    def f(x):
        print("Traced with", x)

    for i in range(5):
        f(2)

    f(3)
    Traced with 2
    Traced with 3
```

Now compare `print` to `tf.print`.

- `tf.print` is graph aware and will run as expected in loops.

Try running the same code where `tf.print()` is added in addition to the regular `print`.

- Note how `tf.print` behaves compared to `print` in graph mode.

```
In [8]: ┌─ @tf.function
    def f(x):
        print("Traced with", x)
        # added tf.print
        tf.print("Executed with", x)

    for i in range(5):
        f(2)

    f(3)
    Traced with 2
    Executed with 2
    Traced with 3
    Executed with 3
```

Avoid defining variables inside the function

This function (not decorated yet) defines a tensor `v` and adds the input `x` to it.

Here, it runs fine.

```
In [9]: ┌─ def f(x):
    v = tf.Variable(1.0)
    v.assign_add(x)
    return v

    print(f(1))
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=2.0>
```

Now if you decorate the function with `@tf.function`.

The cell below will throw an error because `tf.Variable` is defined within the function. The graph mode function should only contain operations.

```
In [10]: ┌─ @tf.function
    def f(x):
        v = tf.Variable(1.0)
        v.assign_add(x)
        return v

    print(f(1))
-----
ValueError                                Traceback (most recent call last)
<ipython-input-10-5729586b3383> in <module>
      5     return v
      6
----> 7 print(f(1))

/opt/conda/lib/python3.7/site-packages/tensorflow/python/eager/def_function.py in __call__(self, *args, **kwargs)
    778         else:
    779             compiler = "nonXla"
--> 780             result = self._call(*args, **kwargs)
    781
    782             new_tracing_count = self._get_tracing_count()

/opt/conda/lib/python3.7/site-packages/tensorflow/python/eager/def_function.py in _call(self, *args, **kwargs)
    838             # Lifting succeeded, so variables are initialized and we can run the
    839             # stateless function.
--> 840             return self._stateless_fn(*args, **kwargs)
    841         else:
    842             canon_args, canon_kwds = \

/opt/conda/lib/python3.7/site-packages/tensorflow/python/eager/function.py in __call__(self, *args, **kwargs)
    2826         """Calls a graph function specialized to the inputs."""
    2827         with self._lock:
--> 2828             graph_function, args, kwargs = self._maybe_define_function(args, kwargs)
    2829             return graph_function._filtered_call(args, kwargs) # pylint: disable=protected-access
    2830
    2831
    2832
    2833
    2834
    2835
    2836
    2837
    2838
    2839
    2840
    2841
    2842
    2843
    2844
    2845
    2846
    2847
    2848
    2849
    2850
    2851
    2852
    2853
    2854
    2855
    2856
    2857
    2858
    2859
    2860
    2861
    2862
    2863
    2864
    2865
    2866
    2867
    2868
    2869
    2870
    2871
    2872
    2873
    2874
    2875
    2876
    2877
    2878
    2879
    2880
    2881
    2882
    2883
    2884
    2885
    2886
    2887
    2888
    2889
    2890
    2891
    2892
    2893
    2894
    2895
    2896
    2897
    2898
    2899
    2900
    2901
    2902
    2903
    2904
    2905
    2906
    2907
    2908
    2909
    2910
    2911
    2912
    2913
    2914
    2915
    2916
    2917
    2918
    2919
    2920
    2921
    2922
    2923
    2924
    2925
    2926
    2927
    2928
    2929
    2930
    2931
    2932
    2933
    2934
    2935
    2936
    2937
    2938
    2939
    2940
    2941
    2942
    2943
    2944
    2945
    2946
    2947
    2948
    2949
    2950
    2951
    2952
    2953
    2954
    2955
    2956
    2957
    2958
    2959
    2960
    2961
    2962
    2963
    2964
    2965
    2966
    2967
    2968
    2969
    2970
    2971
    2972
    2973
    2974
    2975
    2976
    2977
    2978
    2979
    2980
    2981
    2982
    2983
    2984
    2985
    2986
    2987
    2988
    2989
    2990
    2991
    2992
    2993
    2994
    2995
    2996
    2997
    2998
    2999
    3000
    3001
    3002
    3003
    3004
    3005
    3006
    3007
    3008
    3009
    3010
    3011
    3012
    3013
    3014
    3015
    3016
    3017
    3018
    3019
    3020
    3021
    3022
    3023
    3024
    3025
    3026
    3027
    3028
    3029
    3030
    3031
    3032
    3033
    3034
    3035
    3036
    3037
    3038
    3039
    3040
    3041
    3042
    3043
    3044
    3045
    3046
    3047
    3048
    3049
    3050
    3051
    3052
    3053
    3054
    3055
    3056
    3057
    3058
    3059
    3060
    3061
    3062
    3063
    3064
    3065
    3066
    3067
    3068
    3069
    3070
    3071
    3072
    3073
    3074
    3075
    3076
    3077
    3078
    3079
    3080
    3081
    3082
    3083
    3084
    3085
    3086
    3087
    3088
    3089
    3090
    3091
    3092
    3093
    3094
    3095
    3096
    3097
    3098
    3099
    3100
    3101
    3102
    3103
    3104
    3105
    3106
    3107
    3108
    3109
    3110
    3111
    3112
    3113
    3114
    3115
    3116
    3117
    3118
    3119
    3120
    3121
    3122
    3123
    3124
    3125
    3126
    3127
    3128
    3129
    3130
    3131
    3132
    3133
    3134
    3135
    3136
    3137
    3138
    3139
    3140
    3141
    3142
    3143
    3144
    3145
    3146
    3147
    3148
    3149
    3150
    3151
    3152
    3153
    3154
    3155
    3156
    3157
    3158
    3159
    3160
    3161
    3162
    3163
    3164
    3165
    3166
    3167
    3168
    3169
    3170
    3171
    3172
    3173
    3174
    3175
    3176
    3177
    3178
    3179
    3180
    3181
    3182
    3183
    3184
    3185
    3186
    3187
    3188
    3189
    3190
    3191
    3192
    3193
    3194
    3195
    3196
    3197
    3198
    3199
    3200
    3201
    3202
    3203
    3204
    3205
    3206
    3207
    3208
    3209
    3210
    3211
    3212
    3213
    3214
    3215
    3216
    3217
    3218
    3219
    3220
    3221
    3222
    3223
    3224
    3225
    3226
    3227
    3228
    3229
    3230
    3231
    3232
    3233
    3234
    3235
    3236
    3237
    3238
    3239
    3240
    3241
    3242
    3243
    3244
    3245
    3246
    3247
    3248
    3249
    3250
    3251
    3252
    3253
    3254
    3255
    3256
    3257
    3258
    3259
    3260
    3261
    3262
    3263
    3264
    3265
    3266
    3267
    3268
    3269
    3270
    3271
    3272
    3273
    3274
    3275
    3276
    3277
    3278
    3279
    3280
    3281
    3282
    3283
    3284
    3285
    3286
    3287
    3288
    3289
    3290
    3291
    3292
    3293
    3294
    3295
    3296
    3297
    3298
    3299
    3300
    3301
    3302
    3303
    3304
    3305
    3306
    3307
    3308
    3309
    3310
    3311
    3312
    3313
    3314
    3315
    3316
    3317
    3318
    3319
    3320
    3321
    3322
    3323
    3324
    3325
    3326
    3327
    3328
    3329
    3330
    3331
    3332
    3333
    3334
    3335
    3336
    3337
    3338
    3339
    3340
    3341
    3342
    3343
    3344
    3345
    3346
    3347
    3348
    3349
    3350
    3351
    3352
    3353
    3354
    3355
    3356
    3357
    3358
    3359
    3360
    3361
    3362
    3363
    3364
    3365
    3366
    3367
    3368
    3369
    3370
    3371
    3372
    3373
    3374
    3375
    3376
    3377
    3378
    3379
    3380
    3381
    3382
    3383
    3384
    3385
    3386
    3387
    3388
    3389
    3390
    3391
    3392
    3393
    3394
    3395
    3396
    3397
    3398
    3399
    3400
    3401
    3402
    3403
    3404
    3405
    3406
    3407
    3408
    3409
    3410
    3411
    3412
    3413
    3414
    3415
    3416
    3417
    3418
    3419
    3420
    3421
    3422
    3423
    3424
    3425
    3426
    3427
    3428
    3429
    3430
    3431
    3432
    3433
    3434
    3435
    3436
    3437
    3438
    3439
    3440
    3441
    3442
    3443
    3444
    3445
    3446
    3447
    3448
    3449
    3450
    3451
    3452
    3453
    3454
    3455
    3456
    3457
    3458
    3459
    3460
    3461
    3462
    3463
    3464
    3465
    3466
    3467
    3468
    3469
    3470
    3471
    3472
    3473
    3474
    3475
    3476
    3477
    3478
    3479
    3480
    3481
    3482
    3483
    3484
    3485
    3486
    3487
    3488
    3489
    3490
    3491
    3492
    3493
    3494
    3495
    3496
    3497
    3498
    3499
    3500
    3501
    3502
    3503
    3504
    3505
    3506
    3507
    3508
    3509
    3510
    3511
    3512
    3513
    3514
    3515
    3516
    3517
    3518
    3519
    3520
    3521
    3522
    3523
    3524
    3525
    3526
    3527
    3528
    3529
    3530
    3531
    3532
    3533
    3534
    3535
    3536
    3537
    3538
    3539
    3540
    3541
    3542
    3543
    3544
    3545
    3546
    3547
    3548
    3549
    3550
    3551
    3552
    3553
    3554
    3555
    3556
    3557
    3558
    3559
    3560
    3561
    3562
    3563
    3564
    3565
    3566
    3567
    3568
    3569
    3570
    3571
    3572
    3573
    3574
    3575
    3576
    3577
    3578
    3579
    3580
    3581
    3582
    3583
    3584
    3585
    3586
    3587
    3588
    3589
    3590
    3591
    3592
    3593
    3594
    3595
    3596
    3597
    3598
    3599
    3600
    3601
    3602
    3603
    3604
    3605
    3606
    3607
    3608
    3609
    3610
    3611
    3612
    3613
    3614
    3615
    3616
    3617
    3618
    3619
    3620
    3621
    3622
    3623
    3624
    3625
    3626
    3627
    3628
    3629
    3630
    3631
    3632
    3633
    3634
    3635
    3636
    3637
    3638
    3639
    3640
    3641
    3642
    3643
    3644
    3645
    3646
    3647
    3648
    3649
    3650
    3651
    3652
    3653
    3654
    3655
    3656
    3657
    3658
    3659
    3660
    3661
    3662
    3663
    3664
    3665
    3666
    3667
    3668
    3669
    3670
    3671
    3672
    3673
    3674
    3675
    3676
    3677
    3678
    3679
    3680
    3681
    3682
    3683
    3684
    3685
    3686
    3687
    3688
    3689
    3690
    3691
    3692
    3693
    3694
    3695
    3696
    3697
    3698
    3699
    3700
    3701
    3702
    3703
    3704
    3705
    3706
    3707
    3708
    3709
    3710
    3711
    3712
    3713
    3714
    3715
    3716
    3717
    3718
    3719
    3720
    3721
    3722
    3723
    3724
    3725
    3726
    3727
    3728
    3729
    3730
    3731
    3732
    3733
    3734
    3735
    3736
    3737
    3738
    3739
    3740
    3741
    3742
    3743
    3744
    3745
    3746
    3747
    3748
    3749
    3750
    3751
    3752
    3753
    3754
    3755
    3756
    3757
    3758
    3759
    3760
    3761
    3762
    3763
    3764
    3765
    3766
    3767
    3768
    3769
    3770
    3771
    3772
    3773
    3774
    3775
    3776
    3777
    3778
    3779
    3780
    3781
    3782
    3783
    3784
    3785
    3786
    3787
    3788
    3789
    3790
    3791
    3792
    3793
    3794
    3795
    3796
    3797
    3798
    3799
    3800
    3801
    3802
    3803
    3804
    3805
    3806
    3807
    3808
    3809
    3810
    3811
    3812
    3813
    3814
    3815
    3816
    3817
    3818
    3819
    3820
    3821
    3822
    3823
    3824
    3825
    3826
    3827
    3828
    3829
    3830
    3831
    3832
    3833
    3834
    3835
    3836
    3837
    3838
    3839
    3840
    3841
    3842
    3843
    3844
    3845
    3846
    3847
    3848
    3849
    3850
    3851
    3852
    3853
    3854
    3855
    3856
    3857
    3858
    3859
    3860
    3861
    3862
    3863
    3864
    3865
    3866
    3867
    3868
    3869
    3870
    3871
    3872
    3873
    3874
    3875
    3876
    3877
    3878
    3879
    3880
    3881
    3882
    3883
    3884
    3885
    3886
    3887
    3888
    3889
    3890
    3891
    3892
    3893
    3894
    3895
    3896
    3897
    3898
    3899
    3900
    3901
    3902
    3903
    3904
    3905
    3906
    3907
    3908
    3909
    3910
    3911
    3912
    3913
    3914
    3915
    3916
    3917
    3918
    3919
    3920
    3921
    3922
    3923
    3924
    3925
    3926
    3927
    3928
    3929
    3930
    3931
    3932
    3933
    3934
    3935
    3936
    3937
    3938
    3939
    3940
    3941
    3942
    3943
    3944
    3945
    3946
    3947
    3948
    3949
    3950
    3951
    3952
    3953
    3954
    3955
    3956
    3957
    3958
    3959
    3960
    3961
    3962
    3963
    3964
    3965
    3966
    3967
    3968
    3969
    3970
    3971
    3972
    3973
    3974
    3975
    3976
    3977
    3978
    3979
    3980
    3981
    3982
    3983
    3984
    3985
    3986
    3987
    3988
    3989
    3990
    3991
    3992
    3993
    3994
    3995
    3996
    3997
    3998
    3999
    4000
    4001
    4002
    4003
    4004
    4005
    4006
    4007
    4008
    4009
    4010
    4011
    4012
    4013
    4014
    4015
    4016
    4017
    4018
    4019
    4020
    4021
    4022
    4023
    4024
    4025
    4026
    4027
    4028
    4029
    4030
    4031
    4032
    4033
    4034
    4035
    4036
    4037
    4038
    4039
    4040
    4041
    4042
    4043
    4044
    4045
    4046
    4047
    4048
    4049
    4050
    4051
    4052
    4053
    4054
    4055
    4056
    4057
    4058
    4059
    4060
    4061
    4062
    4063
    4064
    4065
    4066
    4067
    4068
    4069
    4070
    4071
    4072
    4073
    4074
    4075
    4076
    4077
    4078
    4079
    4080
    4081
    4082
    4083
    4084
    4085
    4086
    4087
    4088
    4089
    4090
    4091
    4092
    4093
    4094
    4095
    4096
    4097
    4098
    4099
    4100
    4101
    4102
    4103
    4104
    4105
    4106
    4107
    4108
    4109
    4110
    4111
    4112
    4113
    4114
    4115
    4116
    4117
    4118
    4119
    4120
    4121
    4122
    4123
    4124
    4125
    4126
    4127
    4128
    4129
    4130
    4131
    4132
    4133
    4134
    4135
    4136
    4137
    4138
    4139
    4140
    4141
    4142
    4143
    4144
    4145
    4146
    4147
    4148
    4149
    4150
    4151
    4152
    4153
    4154
    4155
    4156
    4157
    4158
    4159
    4160
    4161
    4162
    4163
    4164
    4165
    4166
    4167
    4168
    4169
    4170
    4171
    4172
    4173
    4174
    4175
    4176
    4177
    4178
    4179
    4180
    4181
    4182
    4183
    4184
    4185
    4186
    4187
    4188
    4189
    4190
    4191
    4192
    4193
    4194
    4195
    4196
    4197
    4198
    4199
    4200
    4201
    4202
    4203
    4204
    4205
    4206
    4207
    4208
    4209
    4210
    4211
    4212
    4213
    4214
    4215
    4216
    4217
    4218
    4219
    4220
    4221
    4222
    4223
    4224
    4225
    4226
    4227
    4228
    4229
    4230
    4231
    4232
    4233
    4234
    4235
    4236
    4237
    4238
    4239
    4240
    4241
    4242
```

```

3212     self._function_cache.missed.add(call_context_key)
-> 3213     graph_function = self._create_graph_function(args, kwargs)
3214     self._function_cache.primary[cache_key] = graph_function
3215     return graph_function, args, kwargs

/opt/conda/lib/python3.7/site-packages/tensorflow/python/eager/function.py in _create_graph_function(self, args, kwargs, override_flat_arg_shapes)
    3673         arg_names=arg_names,
    3674         override_flat_arg_shapes=override_flat_arg_shapes,
-> 3675         capture_by_value=self._capture_by_value),
    3676         self._function_attributes,
    3677         function_spec=self.function_spec,

/opt/conda/lib/python3.7/site-packages/tensorflow/python/framework/func_graph.py in func_graph_from_py_func(name, python_func, args, kwargs, signature, func_graph, autograph, autograph_options, add_control_dependencies, arg_names, op_return_value, collections, capture_by_value, override_flat_arg_shapes)
    984     _, original_func = tf_decorator.unwrap(python_func)
--> 986     func_outputs = python_func(*func_args, **func_kwargs)
    987
    988     # invariant: `func_outputs` contains only Tensors, CompositeTensors,
--> 989
--> 990     # wrapped_fn allows AutoGraph to swap in a converted function. We give
--> 991     # the function a weak reference to itself to avoid a reference cycle.
--> 992     return weak_wrapped_fn().__wrapped__(args, **kwargs)
    993     weak_wrapped_fn = weakref.ref(wrapped_fn)
    994
    995     raise e.ag_error_metadata.to_exception(e)
    996
    997     raise

ValueError: in user code:

<ipython-input-10-5729586b3383>:3 f  *
      v=tf.Variable(1.0)
/opt/conda/lib/python3.7/site-packages/tensorflow/python/ops/variables.py:262 __call__  **
      return cls._variable_v2_call(*args, **kwargs)
/opt/conda/lib/python3.7/site-packages/tensorflow/python/ops/variables.py:256 _variable_v2_call
      shape=shape)
/opt/conda/lib/python3.7/site-packages/tensorflow/python/ops/variables.py:67 getter
      return captured_getter(captured_previous, **kwargs)
/opt/conda/lib/python3.7/site-packages/tensorflow/eager/function.py:702 invalid_creator_scope
      "tf.function-decorated function tried to create "

ValueError: tf.function-decorated function tried to create variables on non-first call.

```

To get around the error above, simply move `v = tf.Variable(1.0)` to the top of the cell before the `@tf.function` decorator.

```

In [11]: # define the variables outside of the decorated function
          v = tf.Variable(1.0)

          @tf.function
          def f(x):
              return v.assign_add(x)

          print(f(5))

          tf.Tensor(6.0, shape=(), dtype=float32)

```