1. Tensor.

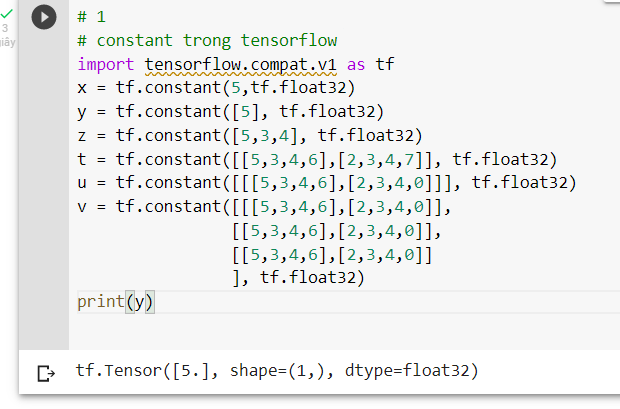
dữ liệu không được lưu trữ dưới dạng số nguyên, số thực hoặc chuỗi. Các giá trị này được đóng gói trong một đối tượng được gọi là tensor

1. Tf.constant()

Tạo 1 tensor

Syntax: tensorflow.constant( value, dtype, shape, name )

VD:

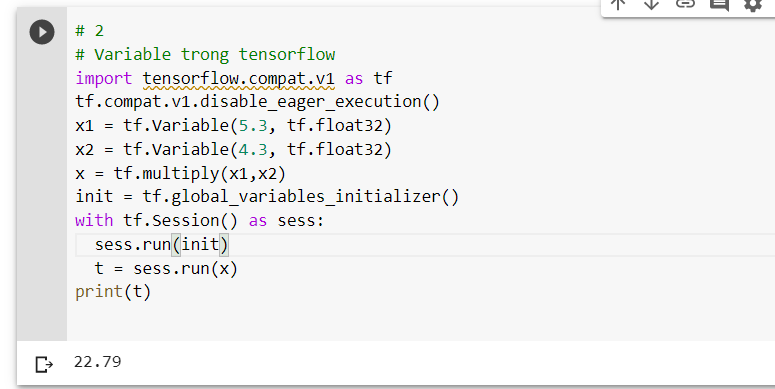


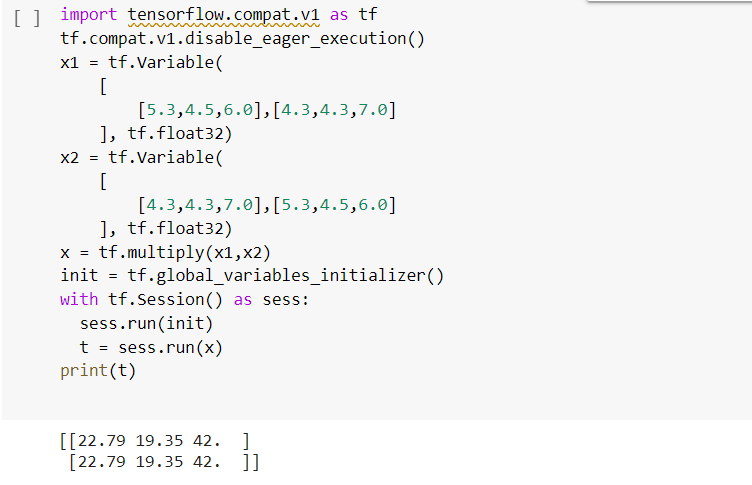
1. Tf.variable()

Tạo 1 tensor

tf.Variable tạo tensor có thể sửa đổi giá trị ban đầu còn tf.constant là bất biến

VD:





1. Tf.placehoder()

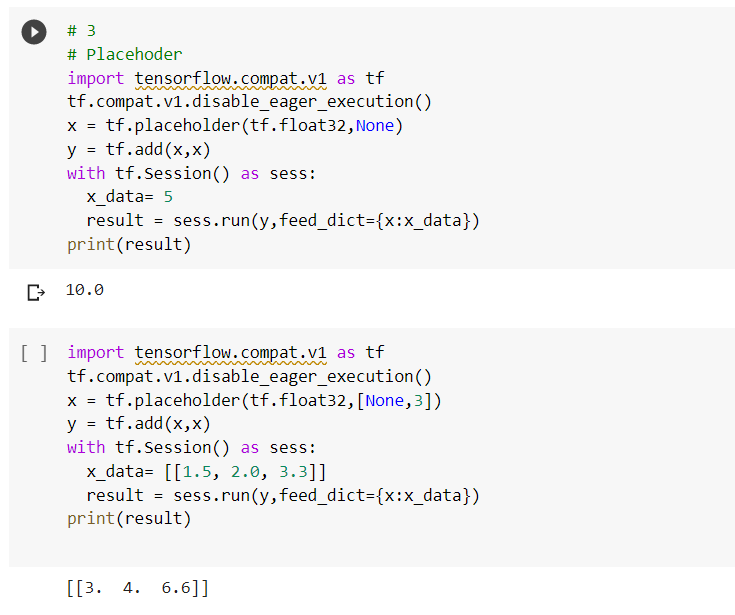
Syntax: tf.compat.v1.placeholder(dtype, shape=None, name=None)

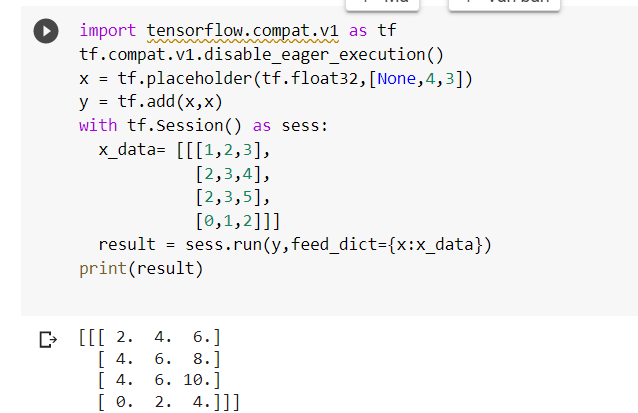
Tạo 1 tensor nhận dữ liệu thay đổi theo thời gian linh hoạt chạy theo phiên

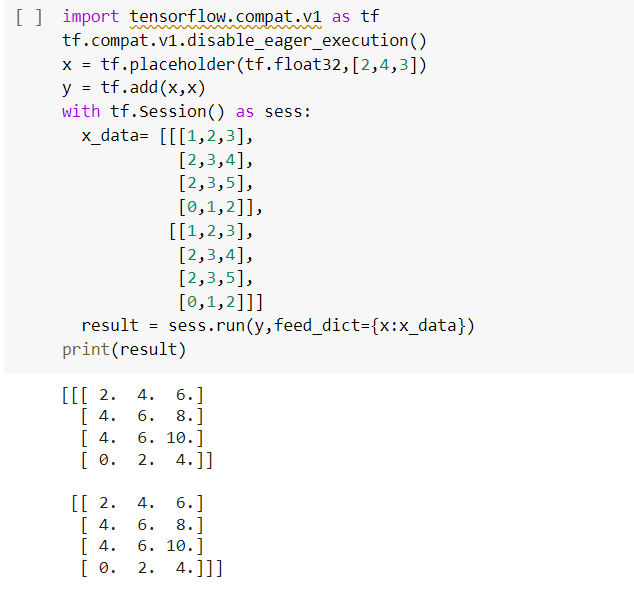
Sử dụng tf.placeholder() và feed\_dict để đặt nhiều hơn 1 tensor

Feed\_dict sẽ lấy dữ liệu khớp với tensor đã khai báo

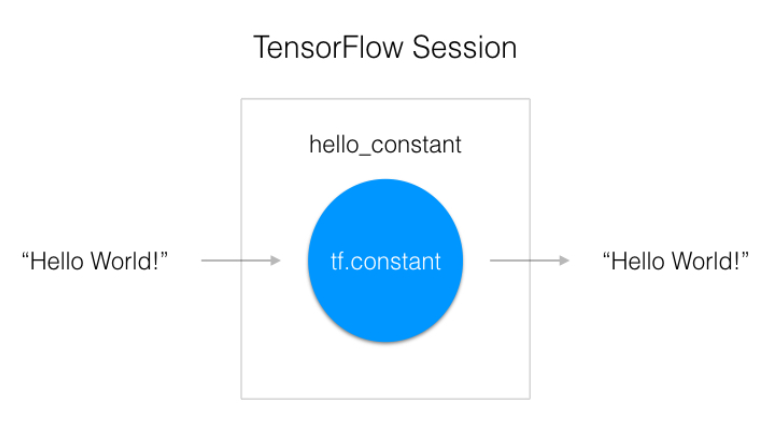
VD:





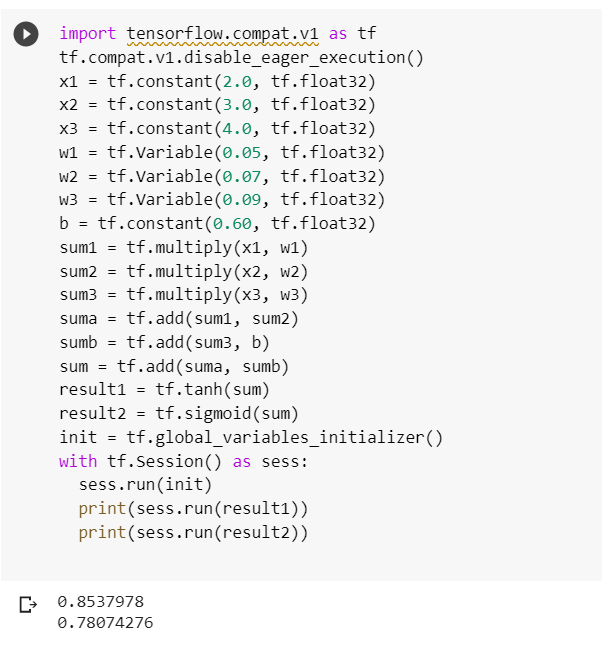


1. Tf.Session()



Một "TensorFlow Session", như được hiển thị ở trên, là một môi trường để chạy một biểu đồ. Phiên phụ trách phân bổ các hoạt động cho (các) GPU và / hoặc (các) CPU, bao gồm cả các máy từ xa

1. Tf.tanh

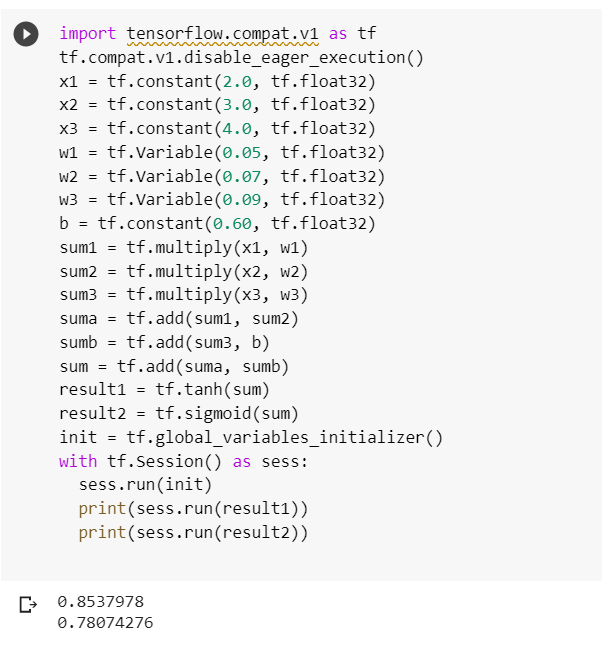


Syntax: tf.nn.tanh(x, name=None) or tf.tanh(x, name=None)

Trả về 1 tensor cùng kiểu với vector ban đầu giá trị từ -1 đến 1



1. Tf.sigmod



Syntax: tf.nn.sigmoid(x, name=None) or tf.sigmoid(x, name=None)

Trả về 1 tensor cùng kiểu với vector ban đầu giá trị từ 0 đến 1



1.a

X1, x2, x3 là các hằng số được đưa vào, mỗi 1 hằng số nhân với 1 giá trị weight sau đấy cộng tất cả lại với bias được giá trị value, tf.tanh ta thu được kết quả mong muốn

Xi tỉ lệ thuận với wi

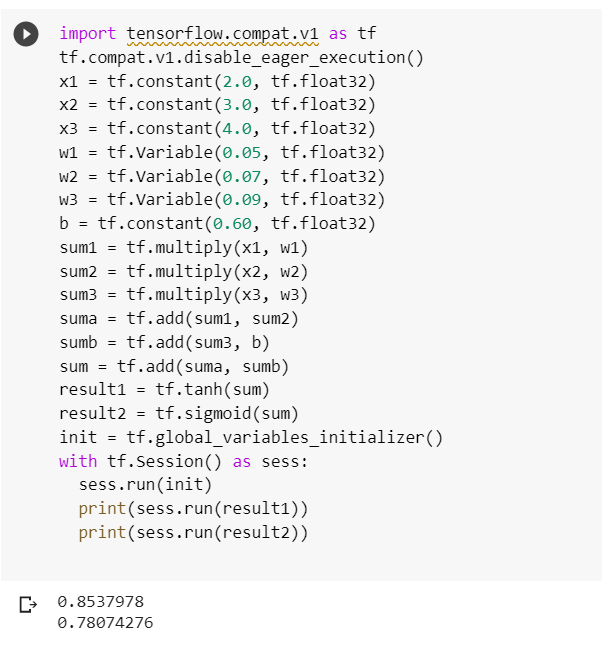
1.b

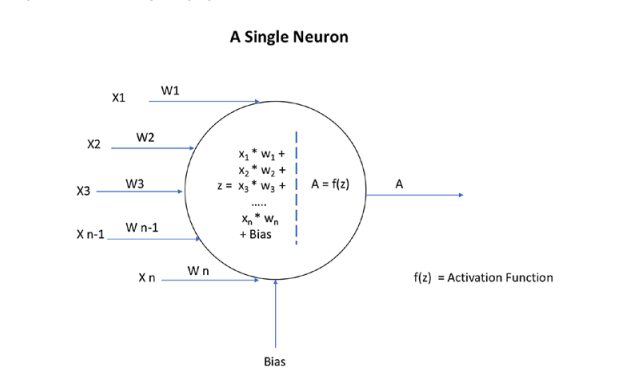
Xi tỉ lệ thuận với wi

A = xW+b

Là hàm tuyến tính với đầu ra là A, W là ma trận trọng số nối 2 lớp. đầu vào x và độ lệch b đều là vector

Hàm tanh chuyển đầu ra thuộc -1 đến 1 tương ứng bao nhiêu phần trăm để phân lớp, dự đoán đầu ra thuộc lớp nào

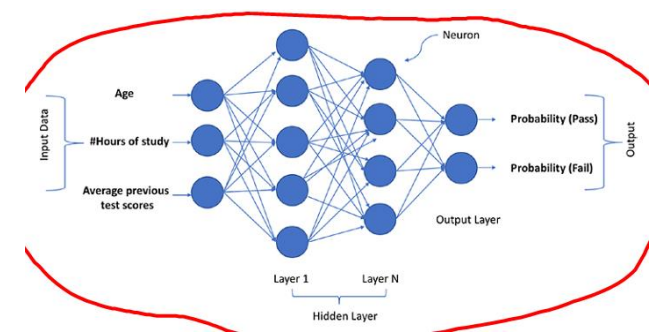




Giải thích: 1 tế bào noron nhận tín hiệu đầu vào từ nhiều tế bào noron khác.

Cụ thể ở đâu là 1 hàm hoạt động dựa trên tổng đầu vào nhân với trọng số tương ứng (fz) phản hồi với 1 giá trị thích hợp dựa trên đầu vào. Đầu vào cao thì ra cao và ngược lại.

Tương tự như tín hiệu kích hoạt( ảnh huwognr cao thì kích hoạt k thì hủy)



**Một tế bào thần kinh nhận một hoặc nhiều đầu vào từ các tế bào thần kinh ở lớp trước**

**Đầu vào Age, hours of study, điểm test trung bình.**

**Đầu ra dự đoán Pass hoặc Fail**

**Feed\_dict: là 1 dict python mapping từ biến tf.placeholder đến data**

**Tf.truncate\_normal() return vector >2**

**Tf.zeros() return tensor 0**

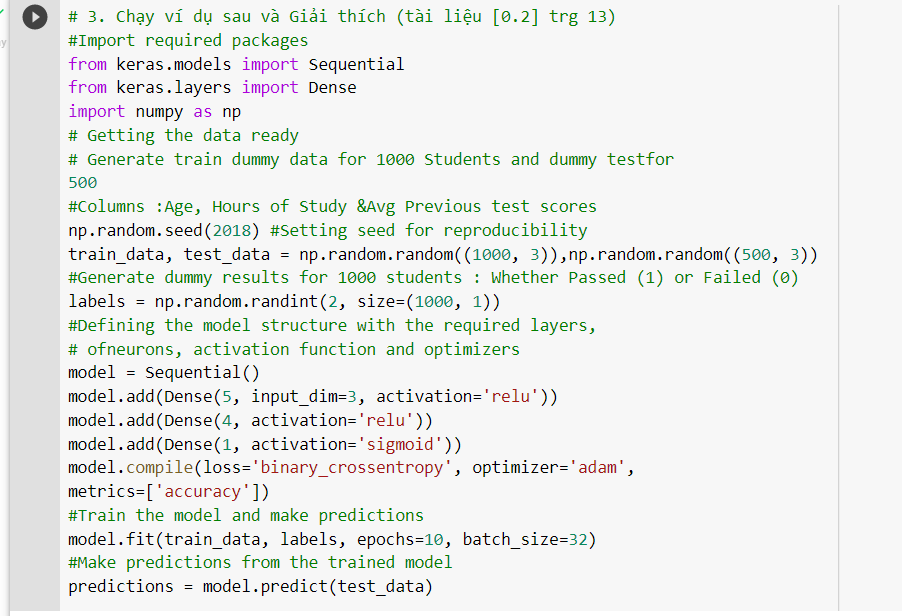
**Tf.softmax() return tensor sum = 1**

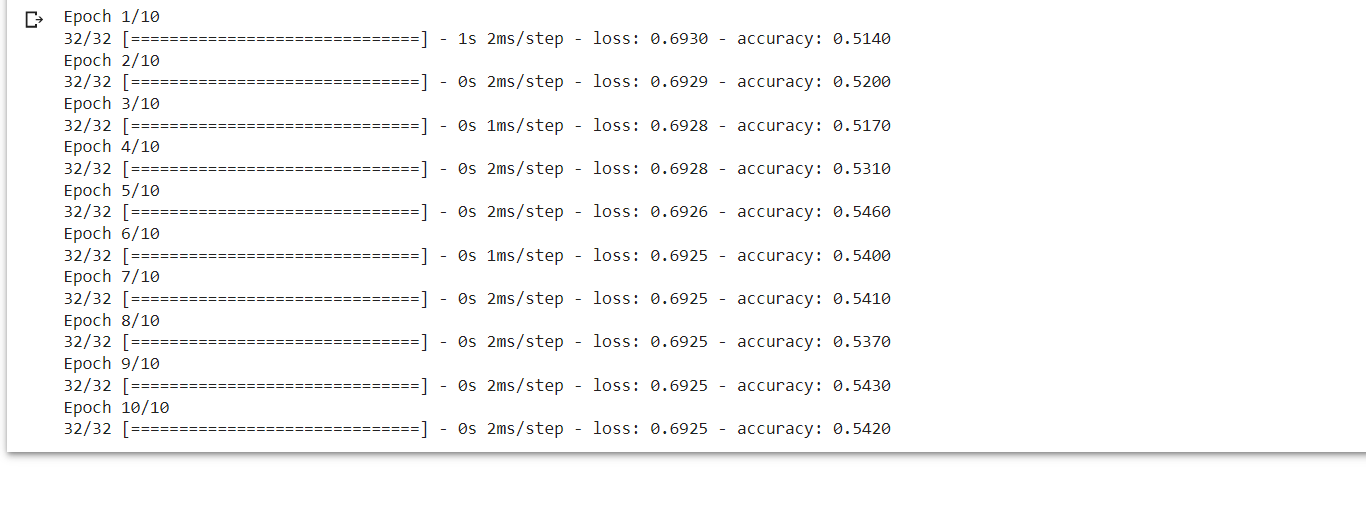
tf.subtract(tf.cast(tf.constant(2.0), tf.int32), tf.constant(1)) # 1

Trừ 2 tensor

tf.cast(tf.constant(2.0), tf.int32) format lại kiểu dữ liệu cho tf.constant

2.





Nó được gọi là “sequential” vì nó liên quan đến việc xác định một lớp Sequential và thêm từng lớp vào mô hình theo cách tuyến tính, từ đầu vào đến đầu ra.

Model.add() để thêm tuần tự các layers

activation : chọn activation như linear, softmax, relu, tanh, sigmoid

Dense ( ): Layer này cũng như một layer neural network bình thường, với các tham số sau:

units : xác định kích thước đầu ra output, như số class sau khi train ( chó , mèo, lợn, gà).

activation : chọn activation đơn giản với sigmoid thì output có 1 class.

use\_bias : có sử dụng bias hay không (True or False)

Hàm compile: Ở hàm này chúng ta sử dụng để training models như thuật toán train qua optimizer như Adam, SGD, RMSprop,..

Hàm fit ():

Bao gồm data train, test đưa vào training.

**Batch\_size** thể hiện số lượng mẫu Mini-batch GD sử dụng cho mỗi lần cập nhật trọng số

**Epoch** là số lần duyệt qua hết số lượng mẫu trong tập huấn luyện.

Giả sử ta có tập huấn luyện gồm 55.000 hình ảnh chọn batch-size là 55 images có nghĩa là mỗi lần cập nhật trọng số, ta dùng 55 images. Lúc đó ta mất 55.000/55 = 1000 iterations (số lần lặp) để duyệt qua hết tập huấn luyện (hoàn thành 1 epochs). Có nghĩa là khi dữ liệu quá lớn, chúng ta không thể đưa cả tập data vào train được, ta phải chia nhỏ data ra thành nhiều batch nhỏ hơn

**Metrics:** Đánh giá mô hình theo tiêu chí nào( accuracy, binary\_accuracy, categorical\_accuracy, cosine\_proximity, clone\_metric)

**Loss:** Dự đoán sai bao nhiêu so với nhãn(mean\_squared\_error, mean\_absolute \_error, hinge, mean\_absolute\_percentage \_error, mean\_squared\_logarithmic\_error, Poisson, binary\_crossentropy ,categorical\_crossentropy)

**Layer:** các khối xây dựng cơ bản trong mạng nơ-ron Keras, 1 hàm đầu vào và 1 hàm đầu ra

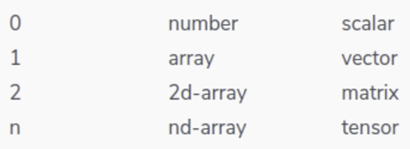
**Optimizer:** Optimizer/loss functions sử dụng để giảm thiểu tổn thất. Trình tối ưu hóa được sử dụng để điều chỉnh trọng số đầu vào, bằng cách so sánh dự đoán và loss functions (SGD (Stochastic Gradient Descent), RMSprop, Adagrad, Adam, Adamax, Nadam)

**Vd:** keras.optimizers.Adam(learning\_rate=0.001)

To compile a Keras model:

model.compile(loss="mean\_squared\_error", optimizer="adam")

**Tensor:** Tổng quát của 1 vector hoặc 1 ma trận n chiều cùng 1 kiểu dữ liệu

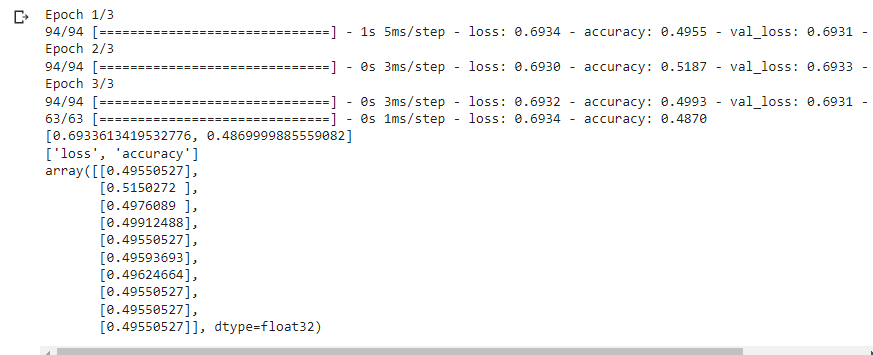


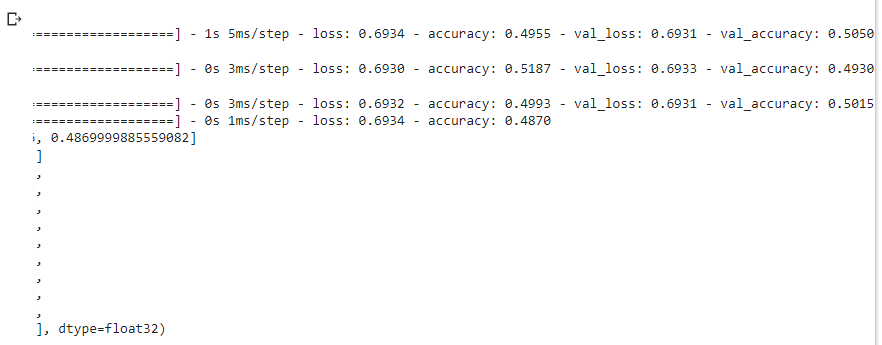
**Weights and Biases (W & B).**

**Weights:** Tham số đầu vào ảnh hưởng đến đầu ra trong mô hình Keras.

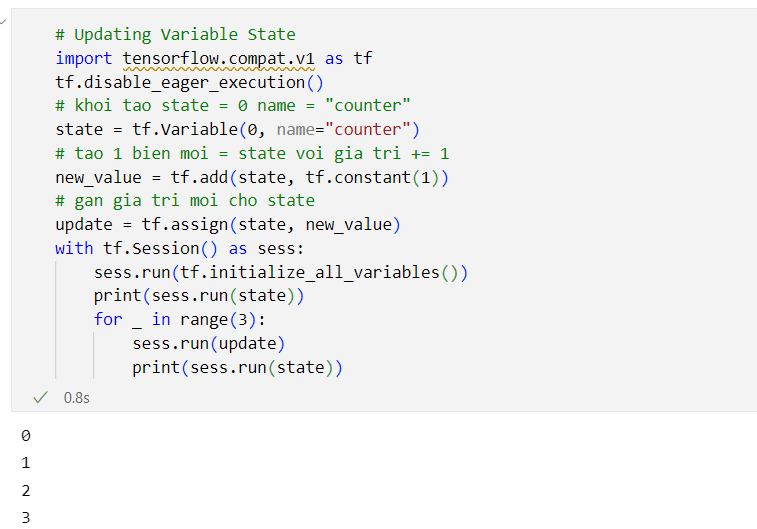
**Biases** : là một giá trị ngưỡng phụ được thêm vào đầu ra.

4.

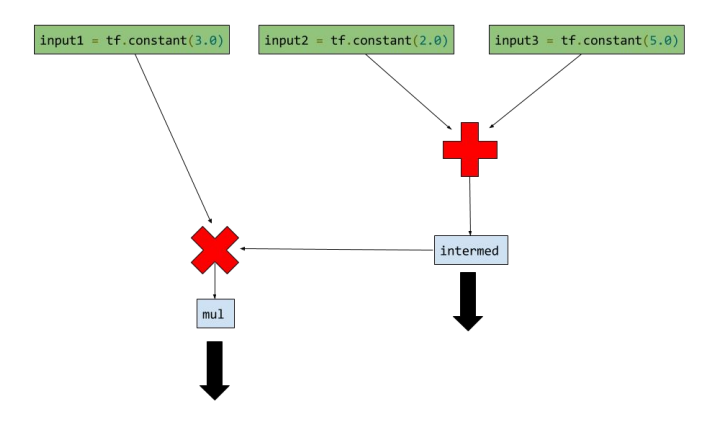


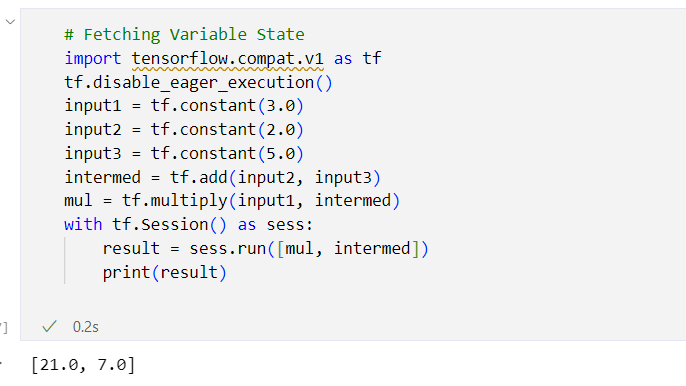


**Updating Variable State**

****

**Fetching Variable State**





2. KERAS