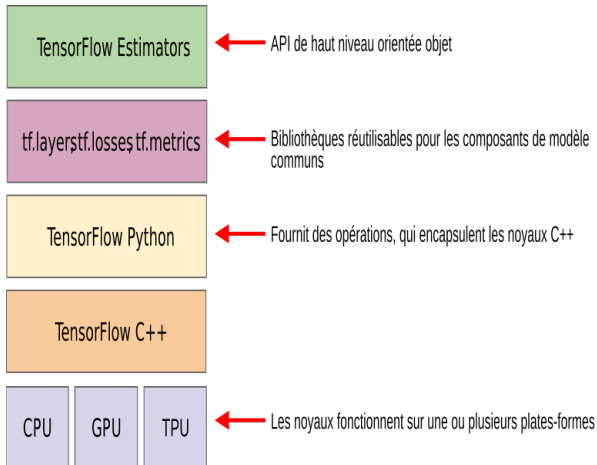
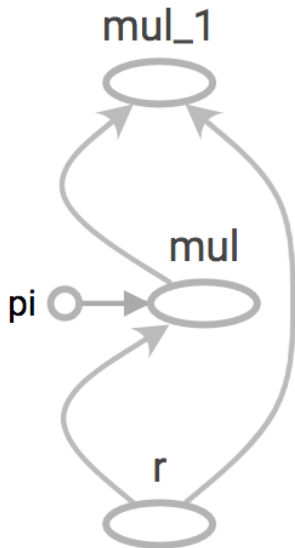




TensorFlow



$$\pi * r^2 \rightarrow$$



Créer un graphe :

```
1 import tensorflow as tf
2 a = tf.constant(6)
3 b = tf.constant(7)
4 mul = tf.multiply(a, b, name='a.b')
5 print(mul)
```

```
1 Tensor("a.b:0", shape=(), dtype=int32)
```

Exécuter un graphe :

```
1 with tf.Session() as session:  
2     print(session.run(mul))
```

```
1 42
```

Créer un graphe tensorflow avec des “placeholder” :

```
1 import tensorflow as tf
2 a = tf.placeholder(tf.float32 , name = "var_a")
3 b = tf.placeholder(tf.float32 , name = "var_b")
4 mul = tf.multiply(a, b, name='a.b')
5 print(mul)
```

```
1 Tensor("a.b:0", dtype=float32)
```

Executer un graphe avec des “placeholder” :

```
1 with tf.Session() as session:  
2     result = session.run(mul, feed_dict={a:[1,2,3], b:[4,5,6]})  
3     print(result)  
4
```

```
1 [ 4. 10. 18.]
```

Matrices et opérations usuelles :

```
1 matrix1 = tf.constant([[1., 2.],[3., 4.]])
2 matrix2 = tf.constant([[5.,6.],[7.,8.]])
3 product = tf.matmul(matrix1, matrix2)
4
5 with tf.Session() as session:
6     result = session.run(product)
7     print(result)
```

```
1 [[19. 22.]
2  [43. 50.]
```


Régression Linéaire (Définition des variables)

```
1  import tensorflow as tf
2
3  # Paramètres du modèle
4  a = tf.Variable(tf.random_normal((1,1)), dtype=tf.float32,name='a')
5  b = tf.Variable(tf.random_normal((1,1)), dtype=tf.float32,name='b')
6
7  # Entrée et sortie du modèle
8  x = tf.placeholder(tf.float32,name="in")
9  y = tf.placeholder(tf.float32,name="out")
10
11 # Données
12 x_train = [1, 2, 3, 4]
13 y_train = [2.99,5.004,6.98,9.001]
```

Régression Linéaire (Définition de la régression et de son erreur) :

```
1  # Le modèle de regression linéaire
2  y_pred=tf.add(tf.multiply(a,x),b,name='y_pred')
3
4  # La fonction de perte (Loss)
5  loss = tf.reduce_sum(tf.square(y_pred - y),name='loss')
6
7  # Méthode de recherche du minimum
8  lr=0.01
9  train = tf.train.GradientDescentOptimizer(lr).minimize(loss)
```

Régression Linéaire :

```
1  # Exécution du graphe 1000 fois sur l'ensemble des données
2  session = tf.Session()
3  init = tf.global_variables_initializer()
4  session.run(init) # reset a et b
5  for i in range(1000):
6      session.run(train, {x: x_train, y: y_train})
7  # Evaluation de l'entraînement du modèle
8  a_app,b_app,erreur=session.run([a,b,loss],{x:x_train,y:y_train})
9  print("a: %s b: %s loss: %s"%(a_app, b_app, erreur))
```

```
1  a: [[2.0009024]] b: [[0.9914936]] loss: 0.00035671948
```

Enregistrer un graphe de calcul tensorflow :

```
1  #Sauvegarde de tout le graphe de calcul
2  sauvegarde=tf.train.Saver()
3  sauvegarde.save(session,'graphs/mygraph')
4
5  !ls -l graphs/
```

```
1  total 40
2  -rw-r--r-- 1 root root    71 Oct  8 12:44 checkpoint
3  -rw-r--r-- 1 root root     8 Oct  8 12:44 mygraph.data-00000-of-00001
4  -rw-r--r-- 1 root root   150 Oct  8 12:44 mygraph.index
5  -rw-r--r-- 1 root root 25739 Oct  8 12:44 mygraph.meta
```

Charger un graphe de calcul tensorflow :

```
1  #Restauration du graphe sauvegardé
2  savedGraph = tf.train.import_meta_graph('graphs/mygraph.meta')
3  #session = tf.Session()
4  savedGraph.restore(session,tf.train.latest_checkpoint('graphs/'))
5
6  #Test du modèle avec de nouvelles valeurs
7  new_x=[5,10,50,100]
8  new_y_pred = session.run(y_pred,feed_dict={x:new_x})
9  print("y_pred(",new_x, ") = ", new_y_pred)
```

```
1  INFO:tensorflow:Restoring parameters from graphs/mygraph
2  y_pred( [5, 10, 50, 100] ) =
3      [[ 10.996005  21.000517 101.03661  201.08173 ]]
```

Réutiliser le graphe pour ajouter de nouveaux calculs :

```
1 g = tf.get_default_graph()
2 y_pred = g.get_tensor_by_name("y_pred:0")
3 #x = g.get_tensor_by_name("in:0")
4 # ...
5
6 #Construction d'un nouveau calcul à partir de y_pred
7 new_op = tf.multiply(y_pred, 10)
8 new_y_pred = session.run(new_op, feed_dict={x:new_x})
9 print("new_op(", new_x, ") = ", new_y_pred)
```

```
1 new_op( [5, 10, 50, 100] ) =
2      [[ 109.96005  210.00517 1010.36615 2010.8173 ]]
```

[Le code présenté, executable dans colaboratory](#)