BÁO CÁO

Họ và tên: Đào Quốc Khánh

MSSV: 2013452

I/ Giới thiệu numpy:

1. Dữ liệu được load vào từ file *abalone.data* và hiển thị ra màn hình

In [2]:
```
# Bai 2-1
data =  pd.read_csv('abalone.data', sep = ",", header = None)
data = data.to_numpy()

print(data)
```

```
[['M' 0.455 0.365 ... 0.101 0.15 15]
 ['M' 0.35 0.265 ... 0.0485 0.07 7]
 ['F' 0.53 0.42 ... 0.1415 0.21 9]
 ...
 ['M' 0.6 0.475 ... 0.2875 0.308 9]
 ['F' 0.625 0.485 ... 0.261 0.296 10]
 ['M' 0.71 0.555 ... 0.3765 0.495 12]]
```

2. Xử lý dữ liệu với trường sex:

In [4]:
```
# Bai 2-2
Sex = data[:, 0]
Sex_M = 0
Sex_F = 0
Sex_I = 0
for i in Sex:
    if i == 'M':
        Sex_M += 1
    elif i == 'F':
        Sex_F += 1
    else:
        Sex_I += 1
print('Length of data: ', len(data))
print('Number of Male: ', Sex_M)
print('Number of Female: ', Sex_F)
print('Number of Infant: ', Sex_I)
```

```
Length of data:  4177
Number of Male:  1528
Number of Female:  1307
Number of Infant:  1342
```

3. Viết hàm để với mỗi dòng dữ liệu trả về dạng {'X': np.array, 't': np.int32} phù hợp với bài toán:

In [5]:
```
# Bai 2-3
def solve(data):
    n = data.shape[0]
    for i in range(n):
        yield data[i, :]

for i in solve(data):
    print(i)
```

```
['M' 0.455 0.365 0.095 0.514 0.2245 0.101 0.15 15]
['M' 0.35 0.265 0.09 0.2255 0.0995 0.0485 0.07 7]
['F' 0.53 0.42 0.135 0.677 0.2565 0.1415 0.21 9]
['M' 0.44 0.365 0.125 0.516 0.2155 0.114 0.155 10]
['I' 0.33 0.255 0.08 0.205 0.0895 0.0395 0.055 7]
['I' 0.425 0.3 0.095 0.3515 0.141 0.0775 0.12 8]
['F' 0.53 0.415 0.15 0.7775 0.237 0.1415 0.33 20]
['F' 0.545 0.425 0.125 0.768 0.294 0.1495 0.26 16]
['M' 0.475 0.37 0.125 0.5095 0.2165 0.1125 0.165 9]
['F' 0.55 0.44 0.15 0.8945 0.3145 0.151 0.32 19]
['F' 0.525 0.38 0.14 0.6065 0.194 0.1475 0.21 14]
['M' 0.43 0.35 0.11 0.406 0.1675 0.081 0.135 10]
['M' 0.49 0.38 0.135 0.5415 0.2175 0.095 0.19 11]
['F' 0.535 0.405 0.145 0.6845 0.2725 0.171 0.205 10]
['F' 0.47 0.355 0.1 0.4755 0.1675 0.0805 0.185 10]
['M' 0.5 0.4 0.13 0.6645 0.258 0.133 0.24 12]
['I' 0.355 0.28 0.085 0.2905 0.095 0.0395 0.115 7]
['F' 0.44 0.34 0.1 0.451 0.188 0.087 0.13 10]
```

4. Viết hàm get_batch(batch_size=16) sử dụng hàm ở 3 để trả về mỗi lần một "batch" với kích thước batch_size. Dạng của output {'X': np.array, 't': np.array}, với shape tương ứng là (batch_size, ?) và (batch_size,)

In [7]:
```
# Bai 2-4
def get_batch(batch_size = 16):
    start, end = 0, batch_size
    total_batch = int(data.shape[0] / batch_size)
    for i in range(total_batch):
        yield solve(data[start:end, :])
        start += batch_size
        end += batch_size

a = get_batch()
for i in a:
    print(list(i))
```

```
[array(['M', 0.455, 0.365, 0.095, 0.514, 0.2245, 0.101, 0.15, 15],
      dtype=object), array(['M', 0.35, 0.265, 0.09, 0.2255, 0.0995, 0.0485, 0.07, 7],
      dtype=object), array(['F', 0.53, 0.42, 0.135, 0.677, 0.2565, 0.1415, 0.21, 9],
      dtype=object), array(['M', 0.44, 0.365, 0.125, 0.516, 0.2155, 0.114, 0.155, 10],
      dtype=object), array(['I', 0.33, 0.255, 0.08, 0.205, 0.0895, 0.0395, 0.055, 7],
      dtype=object), array(['I', 0.425, 0.3, 0.095, 0.3515, 0.141, 0.0775, 0.12, 8],
      dtype=object), array(['F', 0.53, 0.415, 0.15, 0.7775, 0.237, 0.1415, 0.33, 20],
      dtype=object), array(['F', 0.545, 0.425, 0.125, 0.768, 0.294, 0.1495, 0.26, 16],
      dtype=object), array(['M', 0.475, 0.37, 0.125, 0.5095, 0.2165, 0.1125, 0.165, 9],
      dtype=object), array(['F', 0.55, 0.44, 0.15, 0.8945, 0.3145, 0.151, 0.32, 19],
      dtype=object), array(['F', 0.525, 0.38, 0.14, 0.6065, 0.194, 0.1475, 0.21, 14],
      dtype=object), array(['M', 0.43, 0.35, 0.11, 0.406, 0.1675, 0.081, 0.135, 10],
      dtype=object), array(['M', 0.49, 0.38, 0.135, 0.5415, 0.2175, 0.095, 0.19, 11],
      dtype=object), array(['F', 0.535, 0.405, 0.145, 0.6845, 0.2725, 0.171, 0.205, 10],
      dtype=object), array(['F', 0.47, 0.355, 0.1, 0.4755, 0.1675, 0.0805, 0.185, 10],
      dtype=object), array(['M', 0.5, 0.4, 0.13, 0.6645, 0.258, 0.133, 0.24, 12], dtype=object)]
[array(['I', 0.355, 0.28, 0.085, 0.2905, 0.095, 0.0395, 0.115, 7],
      dtype=object), array(['F', 0.44, 0.34, 0.1, 0.451, 0.188, 0.087, 0.13, 10], dtype=object), array(['M', 0.365, 0.295, 0.
08, 0.2555, 0.097, 0.043, 0.1, 7],
```

<u>Kết luận</u>: Qua đây sinh viên đã làm quen cũng như hình dung được cách sử dụng thư viện numpy trong xử lý và tính toán số học, ma trận, mảng,..., xử lý dữ liệu, cũng như tìm hiểu và làm quen với lazy evaluation trong python.

II/ Giới thiệu Tensorflow:

1. Các thao tác cơ bản trên numpy và tensorflow:

   - Các hiện thực hàm đã được hiện thực trong các file code gửi kèm. Sau đây sẽ chỉ đề cập và giải thích các kết quả thu được

   a. Viết hàm matrix_gen(m, n) để sinh ra ma trận các số thực trong khoảng [0, 1] ngẫu nhiên, output là python array biểu diễn cho ma trận. Sử dụng hàm để sinh ra hai ma trận và lưu vào hai biến tương ứng đã cho để sử dụng cho các câu tiếp theo

In [3]:
```
# code sinh viên cho câu a
def matrix_gen(m, n):
    return np.random.random((m, n))

m, n, k = 50, 40, 60
matrix_mn = matrix_gen(m, n)
matrix_nk = matrix_gen(n, k)

print(matrix_mn)
print(matrix_nk)
```

```
[[0.22330281 0.38145557 0.71273891 ... 0.57449327 0.38726355 0.48493111]
 [0.93764155 0.22393999 0.53032504 ... 0.77826778 0.42549767 0.3510201 ]
 [0.5377412  0.7824594  0.91048269 ... 0.28414331 0.12853501 0.58480559]
 ...
 [0.86102287 0.84212208 0.02076622 ... 0.93415734 0.15620593 0.88813706]
 [0.94679756 0.97815129 0.27796104 ... 0.23249903 0.96897704 0.53357472]
 [0.40736721 0.48977564 0.34274774 ... 0.54754696 0.9005337  0.95944327]]
[[0.33540317 0.74390531 0.47886566 ... 0.40972799 0.04091824 0.14298154]
 [0.25746559 0.17869976 0.89718586 ... 0.22363709 0.3454743  0.88677132]
 [0.65886572 0.17678363 0.22312598 ... 0.95295848 0.98296438 0.95111323]
 ...
 [0.20236127 0.5374159  0.75084008 ... 0.63654336 0.94026984 0.64495942]
 [0.85043207 0.76717288 0.25546334 ... 0.36009808 0.40949242 0.70523975]
 [0.18214248 0.51922959 0.53795178 ... 0.90510778 0.14199524 0.59766504]]
```

   b. Viết một hàm py_matrix_mul(matrix_1, matrix_2) để nhân hai ma trận được truyền vào trong đó không sử dụng numpy, tensorflow hay các thư viện khác

```
[[10.04333602  9.72346676  9.15356217 ... 10.50313196  9.62077836
   9.57575014]
 [ 9.6920086   9.91145759 10.29937932 ...  9.92578066 10.06694413
   9.08905957]
 [10.046891    9.98816286 10.22190643 ... 11.29951049  9.96448027
  10.53370855]
 ...
 [10.35551496  9.65445662 11.02806932 ... 12.18337968 11.08790012
  11.06962025]
 [ 9.40223453  9.07298176  9.20582547 ... 10.47146149  8.10587498
   8.82366764]
 [ 7.69268204  8.39625549  8.25929436 ...  9.50640296  8.31740685
   7.67092169]]
Execution time: 82.54837989807129 ms
```

c. Sử dụng numpy để hiện thực cho bài toán nhân hai ma trận với hai ma trân *matrix_mn* và *matrix_nk*, lưu kết quả cuối cùng vào mt_mul_numpy

```
[[10.04333602  9.72346676  9.15356217 ... 10.50313196  9.62077836
   9.57575014]
 [ 9.6920086   9.91145759 10.29937932 ...  9.92578066 10.06694413
   9.08905957]
 [10.046891    9.98816286 10.22190643 ... 11.29951049  9.96448027
  10.53370855]
 ...
 [10.35551496  9.65445662 11.02806932 ... 12.18337968 11.08790012
  11.06962025]
 [ 9.40223453  9.07298176  9.20582547 ... 10.47146149  8.10587498
   8.82366764]
 [ 7.69268204  8.39625549  8.25929436 ...  9.50640296  8.31740685
   7.67092169]]
Execution time: 82.54837989807129 ms
```

d. Sử dụng tensorflow để hiện thực cho câu c thay vì dùng numpy

```
[[10.04333602  9.72346676  9.15356217 ... 10.50313196  9.62077836
   9.57575014]
 [ 9.6920086   9.91145759 10.29937932 ...  9.92578066 10.06694413
   9.08905957]
 [10.046891    9.98816286 10.22190643 ... 11.29951049  9.96448027
  10.53370855]
 ...
 [10.35551496  9.65445662 11.02806932 ... 12.18337968 11.08790012
  11.06962025]
 [ 9.40223453  9.07298176  9.20582547 ... 10.47146149  8.10587498
   8.82366764]
 [ 7.69268204  8.39625549  8.25929436 ...  9.50640296  8.31740685
   7.67092169]]
Execution time: 7.72857666015625 ms
```

e. Với các câu b, c, d, hãy chèn đoạn code để tính thời gian thực thi của mỗi phương pháp và so sánh, đánh giá về mặt thời gian thực thi của các thao tác

- Sau khi chạy thử và tính toán thời gian thực thi của từng phương pháp, ta có:
  + Sử dụng hàm tự viết py_matrix_mul: xấp xỉ 145 - 160 (ms)
  + Sử dụng np.dot trong numpy: xấp xỉ 1 - 2 (ms)
  + Sử dụng tf.matmul (bao gồm cả thao tác chuyển đổi kết quả từ Tenso r về np.array): xấp xỉ 20 - 24 (ms)
  + Sử dụng tf.matmul (không bao gồm cả thao tác chuyển đổi kết quả từ Tensor về np.array): xấp xỉ 2 - 8 (ms)
- Từ đó ta nhận thấy được, việc sử dụng các hàm tích hợp sẵn trong numpy hoặc tensorflow hiệu quả hơn gấp nhiều lần đối với hàm chúng ta tự viết để nhân hai ma trận - O(n^3). Trong đó, việc sử dụng numpy.dot có hiệu quả hơn hẳn so với 2 phương pháp còn lại.

2. Cho đoạn code sinh dữ liệu cho hàm f() như bên dưới. Sinh viên hãy chỉnh sửa đoạn code Linear Regression, chọn các tham số phù hợp để ra được kết quả tốt nhất có thể

- Phần code và kết quả đã được hiện thực và trình bày đầy đủ trong file code đính kèm. Sau đây sẽ là nhận xét về kết quả thu được:

- Sau khi chạy thử và tính toán thời gian thực thi với nhiều bộ số Learning rate và Epoch, ta có các kết quả tiêu biểu về thời gian thực thi và tính phù hợp của mô hình như sau:

+ Learning rate: 0.01, Epoch: 1000 --> 28.91 s, fit không tốt

+ Learning rate: 0.01, Epoch: 1300 --> 34.51 s, fit không tốt

+ Learning rate: 0.01, Epoch: 1700 --> 53.35 s, fit không tốt

+ Learning rate: 0.1, Epoch: 1000 --> 24.03 s, fit tốt

+ Learning rate: 0.5, Epoch: 1000 --> 22.01 s, fit tốt

+ Learning rate: 1, Epoch: 1000 --> 25.17 s, fit tốt

+ Learning rate: 2, Epoch: 200 --> 6.16 s, fit tốt

+ Learning rate: 8, Epoch: 100 --> 2.35 s, fit tốt

+ Learning rate: 10, Epoch: 60 --> 2.12 s, fit tốt

- Với bộ số Learning rate = 10 và Epoch = 60, nếu chúng ta tiếp tục từ từ tăng Learning rate và giảm Epoch, các kết quả thu được (về cả thời gian thực thi và mức độ phù hợp của mô hình) đa phần sẽ không khác biệt nhiều lắm với kết quả thu được hiện tại (tuy nhiên không tăng Learning rate quá mức hay giảm Epoch quá nhỏ vì khi đó thuật toán Gradient Descent sẽ thực hiện không chính xác, không tìm được cực tiểu cũng như không xác định được mô hình).

- Vậy có thể kết luận bộ tham số Learning rate = 10 và Epoch = 60 có thể được coi là một trong những bộ tham số cho ra kết quả "gần như" là tốt nhất.

Kết luận: Qua bài tập này, sinh viên đã nắm kỹ hơn được tác dụng, sự cần thiết cũng như các thao tác căn bản đối với tensorflow. Ngoài ra sinh viên còn thực hành và nắm được những bước căn bản trong việc viết cũng như lựa chọn tham số để tối ưu hóa một chương trình Học Máy đơn giản

III/ Multilayers Perceptron:

- Ở phần này, từ phần hiện thực code lẫn kết quả thu được đều được sinh viên hiện thực và giải thích đầy đủ trong file *2013452.E10_Multilayer_Perceptron.ipynb* gửi kèm nên xin phép không giải thích lại tại đây.
- Ngoài ra, ở bài số 2, với tập dữ liệu cifar_10 ,do những hạn chế về thời gian và phần cứng sử dụng, hiện tại sinh viên chưa thể chọn được các tham số learning rate và epoch phù hợp để mô hình trở nên tốt nhất (nếu được sẽ cập nhật lại các tham số này trong thời gian sớm nhất), cho nên độ chính xác hiện tại của mô hình chỉ vào khoảng:

Accuracy = 0.4389

Kết luận: Ở bài tập này, sinh viên có được những kiến thức cơ bản về một mô hình Multilayers Perceptron (cấu trúc đồ thị, các tham số đầu vào, đầu ra, giải thuật hiện thực,...). Sinh viên cũng đã thực hành kỹ hơn và nắm được rõ hơn về quy trình trong việc viết một chương trình Học Máy cũng như có thêm kinh nghiệm trong xử lý và làm việc với các bộ dữ liệu lớn.