République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
Ecole nationale Supérieure d' Informatique
ex. INI (Institut National de formation en Informatique)

TP Deep learning 3

Réalisé par

- KETFI Raniya SIT1
- KESSI Lamia SIT1

Promotion : 2020/2021

Table des matières

1. Deep learning:

1.1. Définition :

Deep learning ou l'apprentissage profond est un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées de différentes transformations non linéaires.
Les techniques d'apprentissage profond constituent une classe d'algorithmes d'apprentissage automatique qui :

- Utilisent différentes couches d'unité de traitement non linéaire pour l'extraction et la transformation des caractéristiques ; chaque couche prend en entrée la sortie de la précédente ; les algorithmes peuvent être supervisés ou non supervisés, et leurs applications comprennent la reconnaissance de modèles et la classification statistique ;
- Fonctionnent avec un apprentissage à plusieurs niveaux de détail ou de représentation des données ; à travers les différentes couches, on passe de paramètres de bas niveau à des paramètres de plus haut niveau, où les différents niveaux correspondent à différents niveaux d'abstraction des données.

Ces architectures permettent aujourd'hui de conférer du « sens » à des données en leur donnant la forme d'images, de sons ou de textes.
L'apprentissage profond utilise des couches cachées de réseaux de neurones artificiels, des « machines de Boltzmann restreintes », et des séries de calculs propositionnels complexes. Les algorithmes d'apprentissage profond s'opposent aux algorithmes d'apprentissage peu profonds du fait du nombre de transformations réalisées sur les données entre la couche d'entrée et la couche de sortie, où une transformation correspond à une unité de traitement définie par des poids et des seuils.

1.2. L'objectif de deep learning:

L'idée du Deep Learning est de construire automatiquement cette représentation pertinente des données à travers la phase d'apprentissage, évitant ainsi une intervention humaine. On parle donc d'apprentissage par représentation. Un algorithme de Deep Learning va apprendre des représentations hiérarchiques de plus en plus complexes de données. Ce type d'algorithme est donc adapté aux données de signaux (Images, textes, sons,…), car, par essence, celles-ci sont très hiérarchiques.
Donc le deep learning :

- Répondre à des questions du type "que peut-on déduire de ces données ?"
- N'identifie pas les caractéristiques essentielles du traitement par un humain dans algorithme préalable, mais directement par l'algorithme de Deep Learning.

2. Les caractéristiques du dataset WISDM (Smartphone and Smartwatch Activity and Biometrics Dataset):

Pour ce projet, les données sont extraites du site Wireless Sensor Data Mining Lab (WISDM) qui contient un fichier texte brut avec 6 autres attributs et environ 1098207 échantillons (lignes) d'activités différentes effectuées par les utilisateurs. Ces données sont collectées à partir de 36 individus, dont chacun a été invité à effectuer 6 activités de 3 minutes chacune. Chaque sujet avait un smart-montre placée sur sa main dominante et un smartphone dans leur poche. La collecte de données était contrôlée par une application personnalisée exécutée sur le smartphone et smart Watch. Les données du capteur qui a été recueilli était de l'accéléromètre et g y- corde sur le smartphone et la smartwatch, donnant quatre capteurs au total. Ces données ont été collectées à une fréquence de 20 Hz (toutes les 50 ms).
Voici une table qui contient les caractéristiques générales du dataset :

Sommaires des informations	
Nombre d'individus	**36 utilisateurs**
Nombre d'activités	**6 activités**
Durée de chaque activité	
Fréquence de capteur	**20 Hz**
Smartphone utilisé	**Google Nexus 5/5x** **Samsung Galaxy S5**
Smart-Watch utilisée	**LG G Watch**
Nombre de lignes	**1098207**
Nombre de valeurs manquantes	**Pas de valeur manquante**

Les activités représentées dans le dataset sont: « walking », « standing », « sitting », « jogging », « upstairs », « downstairs ». Pour chaque activité on a 3 axes de mesures pour chaque capteur en plus de timestamp.
La structures du dataset est comme suit:
Identifiant de l'utilisateur , Nom d'activité,Timestamp, x_axis, y_axis, z_axis;

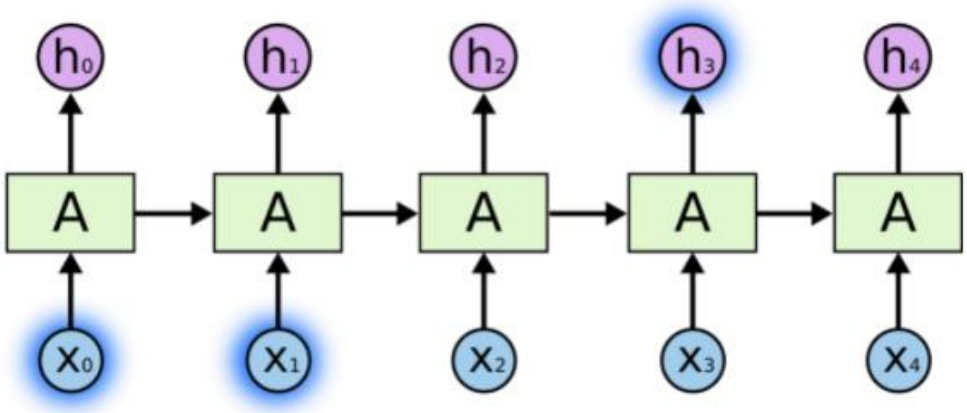
Champ	Description
L'id d'utilisateur	Un identifiant de type entier de 1 à 36
Activité	Nom de l'activité de type objet (int64)
Timestamp	Type entier (int64)
X	Type real (float64), peut être positif ou négatif
Y	Type real (float64), peut être positif ou négatif
Z	Type real (float64), peut être positif ou négatif

La distribution des classes:
Walking -> 424,400 -> 38.6%,
Jogging -> 342,177 -> 31.2%,
Upstairs -> 122,869 -> 11.2%,
Downstairs -> 100,427 -> 9.1%,
Sitting -> 59,939 -> 5.5%,
Standing -> 48,395 -> 4.4%.

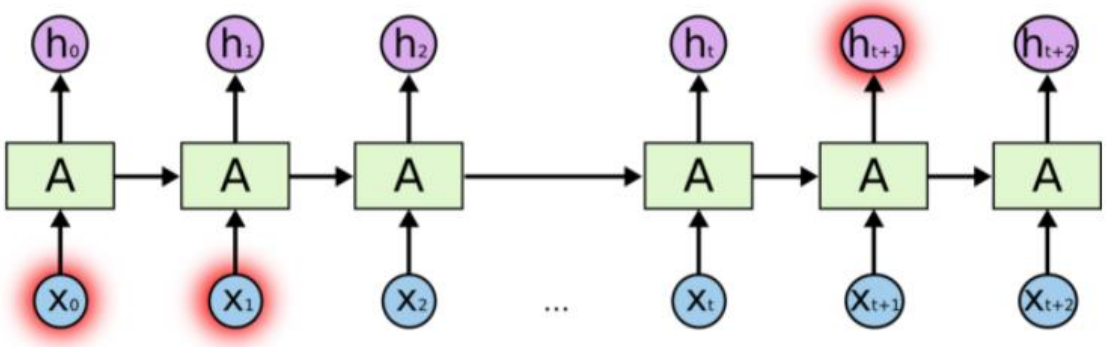
3. LSTM (Long short time memory):

3.1. Architecture LSTM:

L'un des attraits des RNN est l'idée qu'ils pourraient être en mesure de connecter des informations précédentes à la tâche actuelle, comme l'utilisation d'images vidéo précédentes pourrait éclairer la compréhension de l'image actuelle. Si les RNN pouvaient faire cela, ils seraient extrêmement utiles. Parfois, nous n'avons besoin que de regarder des informations récentes pour effectuer la tâche actuelle. Prenant l'exemple d'un modèle d'une langue essayant de prédire le mot suivant en fonction des précédents.Dans le cas ou l'écart entre l'information pertinente et leur besoin est petit, les RNN peuvent apprendre à utiliser les informations passées.

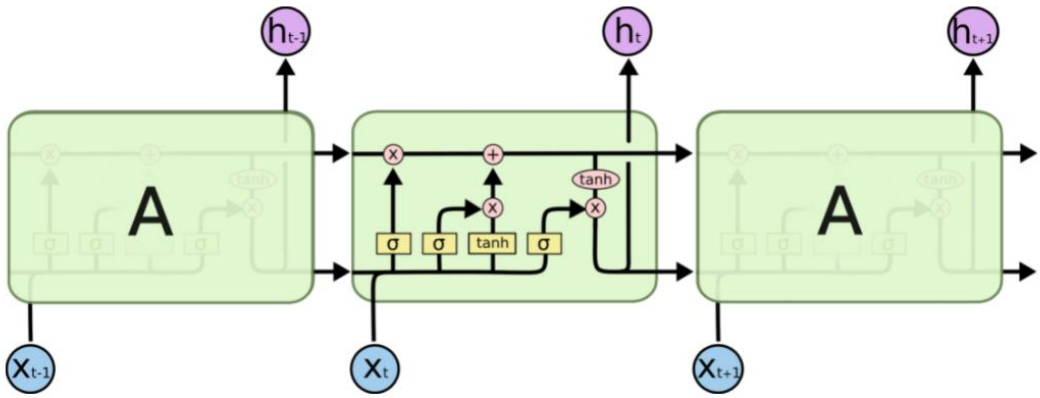


Mais il y a aussi des cas où nous avons besoin de plus de contexte pour prédire, Il est tout à fait possible que l'écart entre les informations pertinentes et le point où elles sont nécessaires devienne très grand.Malheureusement, à mesure que cet écart se creuse, les RNN deviennent incapables d'apprendre à connecter les informations.

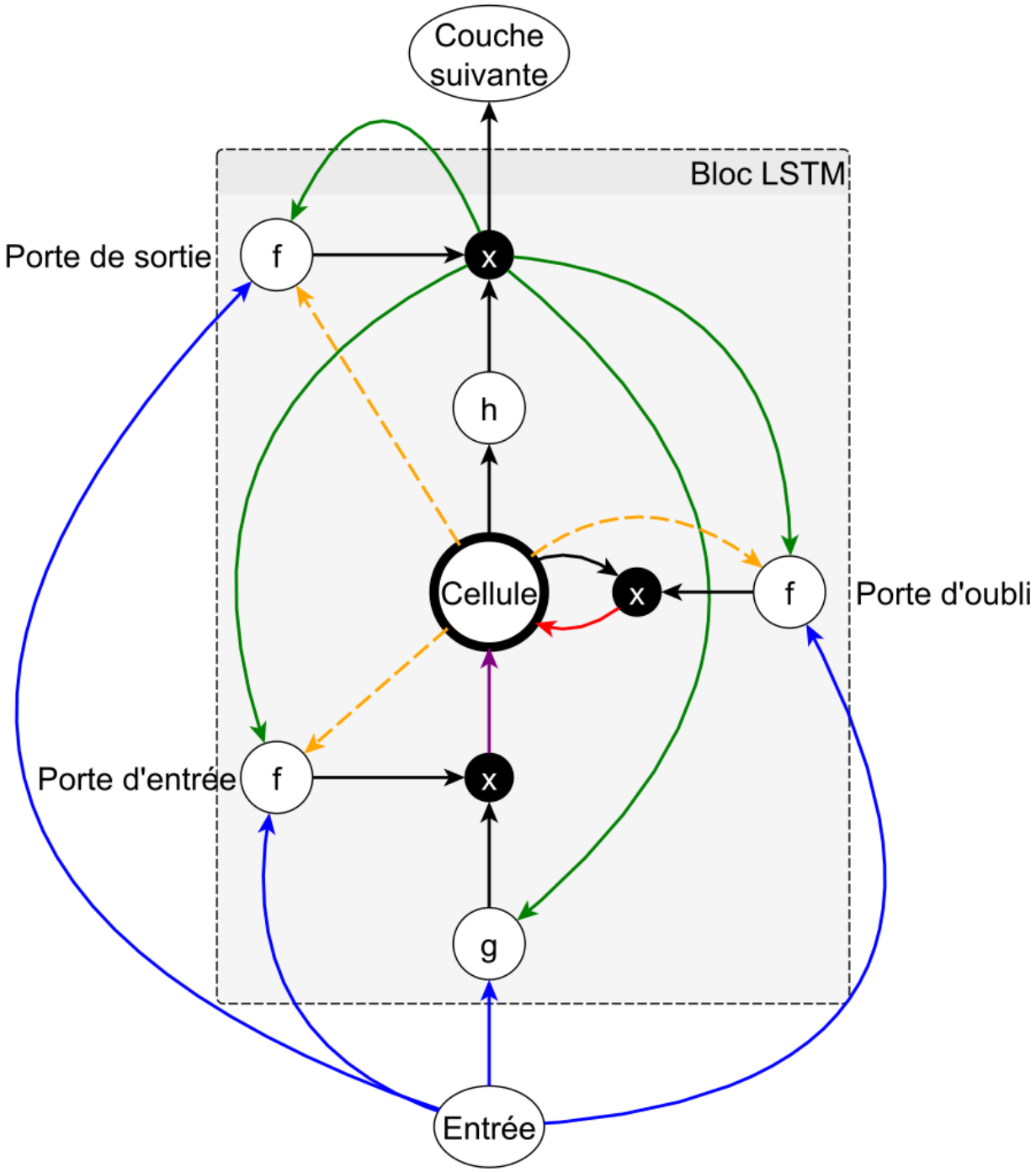


Afin de modéliser des dépendances à très long terme, il est nécessaire de donner aux réseaux de neurones récurrents la capacité de maintenir un état sur une longue période de temps. C'est le but des cellules LSTM (Long Short Term Memory), qui possèdent une mémoire interne appelée *cellule* (ou *cell*). La cellule permet de maintenir un état aussi longtemps que nécessaire.

Les LSTM ont également cette structure en chaîne comme les RNN, mais le module répétitif a une structure différente. Au lieu d'avoir une seule couche de réseau neuronal, il y en a quatre, qui interagissent d'une manière très spéciale.



3.2. La structure d'une cellule LSTM :



Comme on peut le constater sur le schéma, la cellule mémoire peut être pilotée par **trois portes** de contrôle qu'on peut voir comme des vannes :

- **La porte d'entrée** décide si l'entrée doit **modifier le contenu** de la cellule ;
- **La porte d'oubli** décide s'il faut **remettre à 0** le contenu de la cellule ;
- **La porte de sortie** décide si le contenu de la cellule doit **influer sur la sortie** du neurone.

4. Analyser les résultats:

4.1. Importations des bibliotheques :

Dans ce projet on besoin de tensorflow et keras qui supporte les algorithmes de deep learning, ainsi que les bibliotheques pandas pour la manipulation et l'analyse des données et numpy pour manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux

In [1]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from pandas.plotting import register_matplotlib_converters
```

4.2. Chargement du dataset :

Le fichier WISDM_ar_v1_raw n'a pas de noms de colonne. De plus, la dernière colonne contient un ";" après chaque valeur. Corrigeons ce problème:

In [4]:
```
column_names = ['user_id', 'activity', 'timestamp', 'x_axis', 'y_axis', 'z_axis']

df = pd.read_csv('WISDM_ar_v1.1_raw.txt', header=None, names=column_names)
df.z_axis.replace(regex=True, inplace=True, to_replace=r';', value=r'')
df['z_axis'] = df.z_axis.astype(np.float64)
df.dropna(axis=0, how='any', inplace=True)
```

In [5]:
```
df.head()
```

Out[5]:

	user_id	activity	timestamp	x_axis	y_axis	z_axis
0	33	Jogging	49105962326000	-0.694638	12.680544	0.503953
1	33	Jogging	49106062271000	5.012288	11.264028	0.953424
2	33	Jogging	49106112167000	4.903325	10.882658	-0.081722
3	33	Jogging	49106222305000	-0.612916	18.496431	3.023717
4	33	Jogging	49106332290000	-1.184970	12.108489	7.205164

Les données présentent les caractéristiques suivantes:

- user_id – identifiant unique de chaque individus effectuant une
- activity – le nom de l'activité
- timestamp
- x_axis, y_axis, z_axis – la valeur de l'accéléromètre pour chaque axe

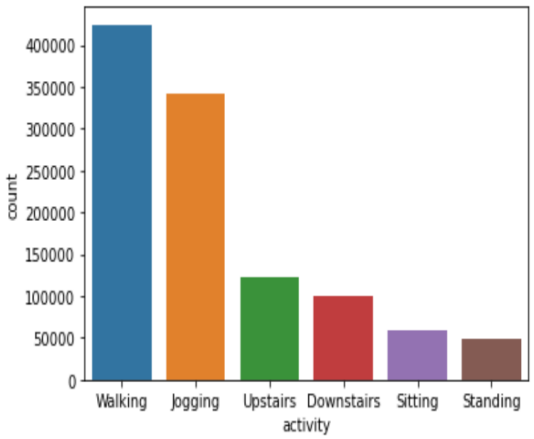
```
In [6]: df.shape
Out[6]: (1098203, 6)
```

Donc on a 1098203 lignes dans le fichier et 6 colonnes.

4.3. L'exploration du dataset :

D'abord on va visualiser la distribution des classes :

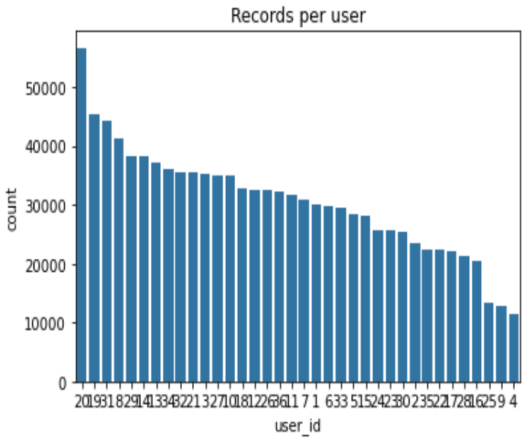
```
In [7]: sns.countplot(x = 'activity',
                      data = df,
                      order = df.activity.value_counts().index);
```



Donc on remarque que les classes sont déséquilibrées, walking et jogging sont surreprésentées.

Maintenant on veut savoir le nombre de données par individus :

```
In [8]: sns.countplot(x = 'user_id',
                      data = df,
                      palette=[sns.color_palette()[0]],
                      order = df.user_id.value_counts().index);
plt.title("Records per user");
```

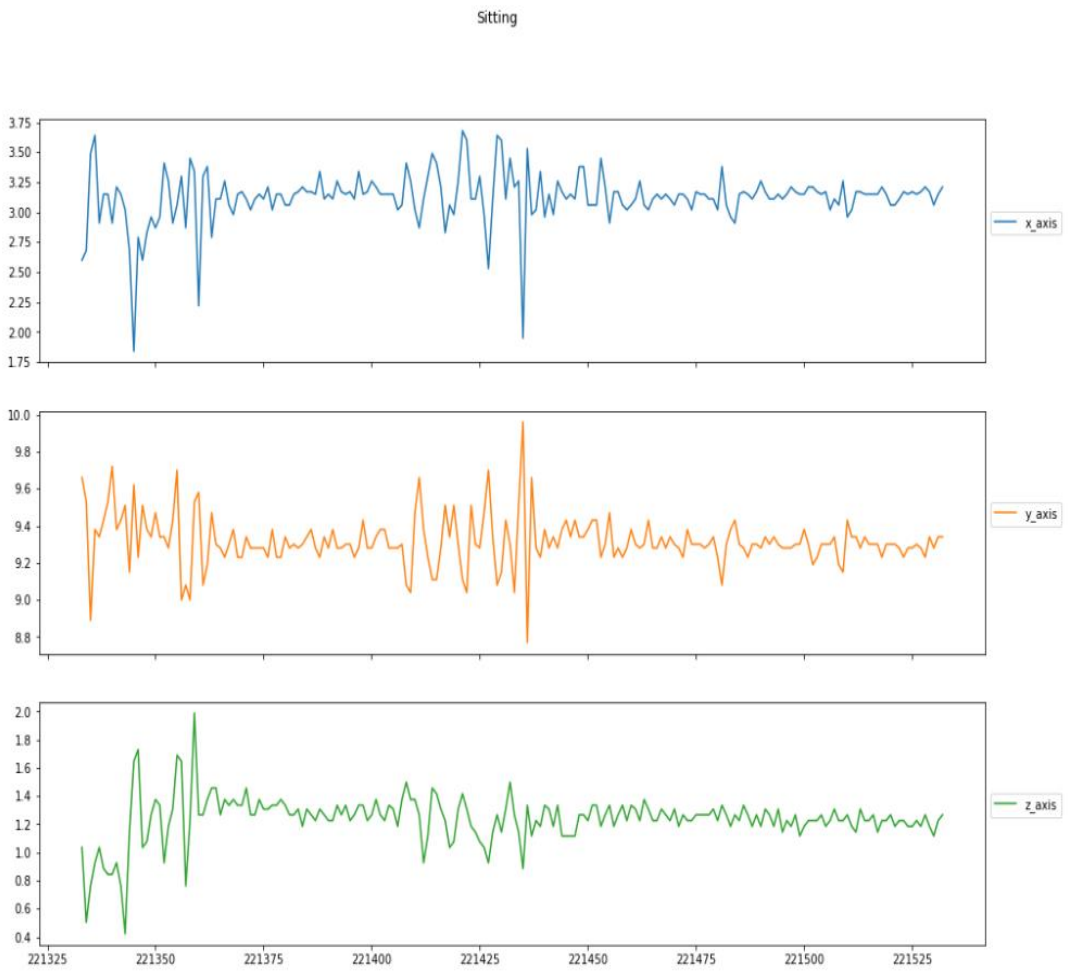


La plupart des individus ont un bon nombre d'échantillons sur leurs activités (à part les 3 derniers).

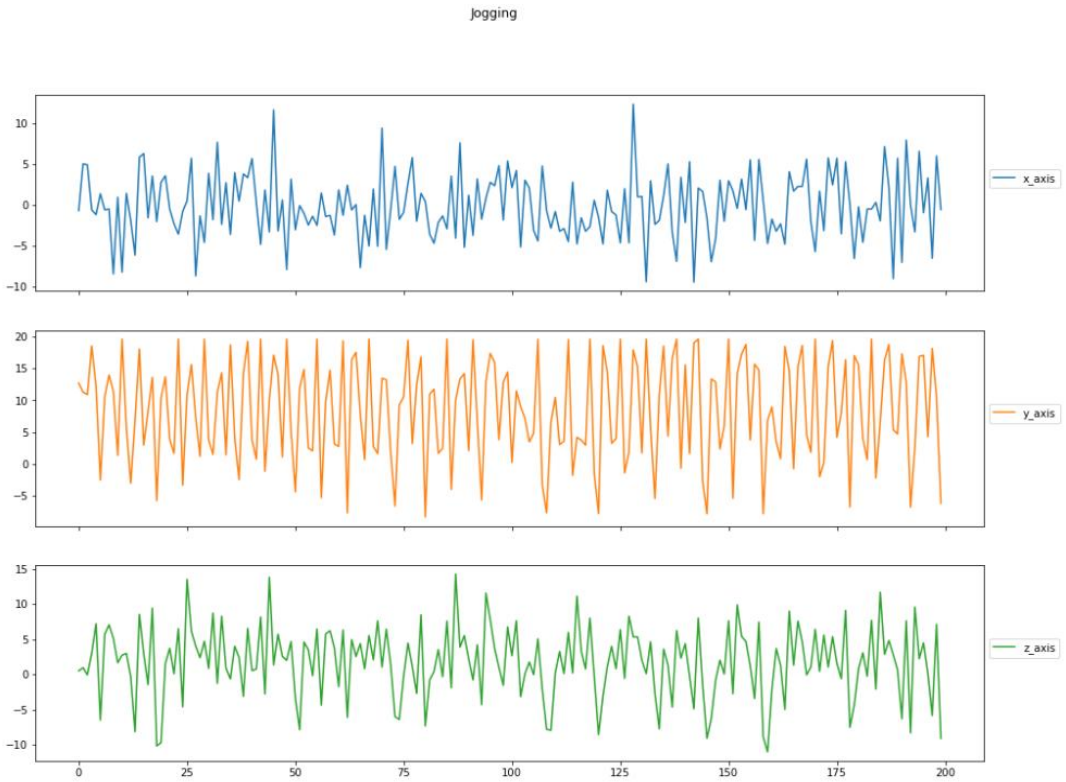
Maintenant on va visualiser les cordonnées des activités selon les 3 axes x, y et z, pour cela on va prendre les 200 premiers enregistrements :

```
In [75]: def plot_activity(activity, df):
             data = df[df['activity'] == activity][['x_axis', 'y_axis', 'z_axis']][:200]
             axis = data.plot(subplots=True, figsize=(16, 12),
                              title=activity)
             for ax in axis:
                 ax.legend(loc='lower left', bbox_to_anchor=(1.0, 0.5))
```

```
In [10]: plot_activity("Sitting", df);
```



```
In [11]: plot_activity("Jogging", df);
```



On remarque que les données peuvent être séparées et classifiées.

4.4. Prétraitement de données :

La première chose à faire est de diviser les données en ensembles de données d'entraînement et de test. Nous utiliserons les données des utilisateurs dont l'ID est inférieur ou égal à 30 comme des donnes d'entrainement (80%). Le reste sera destiné aux test (20%) :

```
In [12]: df_train = df[df['user_id'] <= 30]
         df_test = df[df['user_id'] > 30]
```

Ensuite, nous allons mettre à l'échelle les valeurs des données de l'accéléromètre:

```
In [13]: from sklearn.preprocessing import RobustScaler

         scale_columns = ['x_axis', 'y_axis', 'z_axis']

         scaler = RobustScaler()

         scaler = scaler.fit(df_train[scale_columns])

         df_train.loc[:, scale_columns] = scaler.transform(df_train[scale_columns].to_numpy())
         df_test.loc[:, scale_columns] = scaler.transform(df_test[scale_columns].to_numpy())
         df_test
```

Voici les données centrées :

Out[13]:

	user_id	activity	timestamp	x_axis	y_axis	z_axis
0	33	Jogging	49105962326000	-0.146278	0.596472	0.118522
1	33	Jogging	49106062271000	0.602662	0.426830	0.209481
2	33	Jogging	49106112167000	0.588363	0.381157	0.000000
3	33	Jogging	49106222305000	-0.135553	1.292986	0.628442
4	33	Jogging	49106332290000	-0.210626	0.527963	1.474634
...	...	...	...	...	...	...
832892	31	Standing	25207361641000	1.020997	-0.380838	0.046893
832893	31	Standing	25207401497000	1.120735	-0.440719	0.311996
832894	31	Standing	25207441536000	0.990814	-0.353293	0.139983
832895	31	Standing	25207521553000	0.996063	-0.348503	0.156172
832896	31	Standing	25207561622000	1.040682	-0.390419	0.235096

199843 rows × 6 columns

Maintenant on doit créer les séquencés à partir des données centrées. Pour cela on va modifier la fonction create_dataset .

Nous choisissons l'étiquette (catégory) en utilisant le mode de toutes les catégories de la séquence. Autrement dit, étant donné une séquence de longueur time_steps, nous la classons comme la catégorie qui se produit le plus souvent.

In [14]:
```
from scipy import stats

def create_dataset(X, y, time_steps=1, step=1):
    Xs, ys = [], []
    for i in range(0, len(X) - time_steps, step):
        v = X.iloc[i:(i + time_steps)].values
        labels = y.iloc[i: i + time_steps]
        Xs.append(v)
        ys.append(stats.mode(labels)[0][0])
    return np.array(Xs), np.array(ys).reshape(-1, 1)

TIME_STEPS = 200
STEP = 5

X_train, y_train = create_dataset(
    df_train[['x_axis', 'y_axis', 'z_axis']],
    df_train.activity,
    TIME_STEPS,
    STEP
)

X_test, y_test = create_dataset(
    df_test[['x_axis', 'y_axis', 'z_axis']],
    df_test.activity,
    TIME_STEPS,
    STEP
)
```

Voici la nouvelle séquence, nous avons considérablement réduit la quantité de données d'entraînement et de test :

In [15]:
```
print(X_train.shape, y_train.shape)
```

```
(179632, 200, 3) (179632, 1)
```

La dernière étape de prétraitement est l'encodage des catégories, Cela crée une colonne binaire pour chaque catégorie et renvoie une matrice creuse ou un tableau dense:

In [16]:
```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore', sparse=False)

enc = enc.fit(y_train)

y_train = enc.transform(y_train)
y_test = enc.transform(y_test)
```

In [17]:
```
print(X_train.shape, y_train.shape)
```

```
(179632, 200, 3) (179632, 6)
```

4.5. Classification :

On va appliquer un model LSTM bidirectionnel, avec 4 couches.

Units : Entier positif, dimensionnalité de l'espace de sortie.

Metrics : Liste des métriques à évaluer par le modèle lors de la formation et des tests.

Loss : Le but des fonctions de perte est de calculer la quantité qu'un modèle doit chercher à minimiser pendant l'entraînement.

Rate :

Epochs :nombre d'itération effectuer sur les donnes d'entrainement pour entrainer le model.

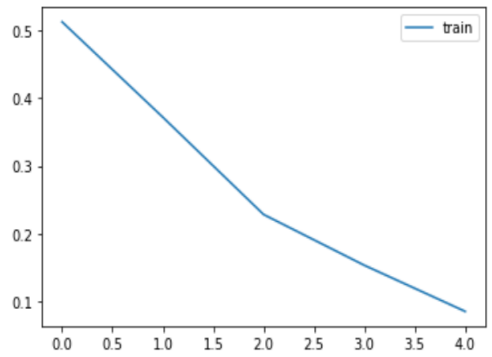
In [71]:
```
model = keras.Sequential()
model.add(
      keras.layers.Bidirectional( keras.layers.LSTM(
          units=128,
          input_shape=[X_train.shape[1], X_train.shape[2]]
      )

))
model.add(keras.layers.Dropout(rate=0.5))
model.add(keras.layers.Dense(units=128, activation='relu'))
model.add(keras.layers.Dense(y_train.shape[1], activation='softmax'))
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['acc']
)
```

In [72]:
```
history=model.fit( X_train, y_train,epochs =5,batch_size=32)
```

```
Epoch 1/5
5614/5614 [==============================] - 1378s 240ms/step - loss: 0.5120 - acc: 0.8142
Epoch 2/5
5614/5614 [==============================] - 1356s 241ms/step - loss: 0.3717 - acc: 0.8614
Epoch 3/5
5614/5614 [==============================] - 1326s 236ms/step - loss: 0.2284 - acc: 0.9136
Epoch 4/5
5614/5614 [==============================] - 1326s 236ms/step - loss: 0.1536 - acc: 0.9474
Epoch 5/5
5614/5614 [==============================] - 1340s 239ms/step - loss: 0.0858 - acc: 0.9723
```

Voici comment s'est déroulé le processus d'entrainement:

In [73]:
```
plt.plot(history.history['loss'], label='train')
plt.legend();
```



Donc si on applique le model au données de test on aura le résultats suivant :

In [75]:
```
model.evaluate(X_test, y_test)
```

```
1248/1248 [==============================] - 110s 88ms/step - loss: 0.4386 - acc: 0.8975
```

Out[75]:
```
[0.4386385381221771, 0.8975180983543396]
```

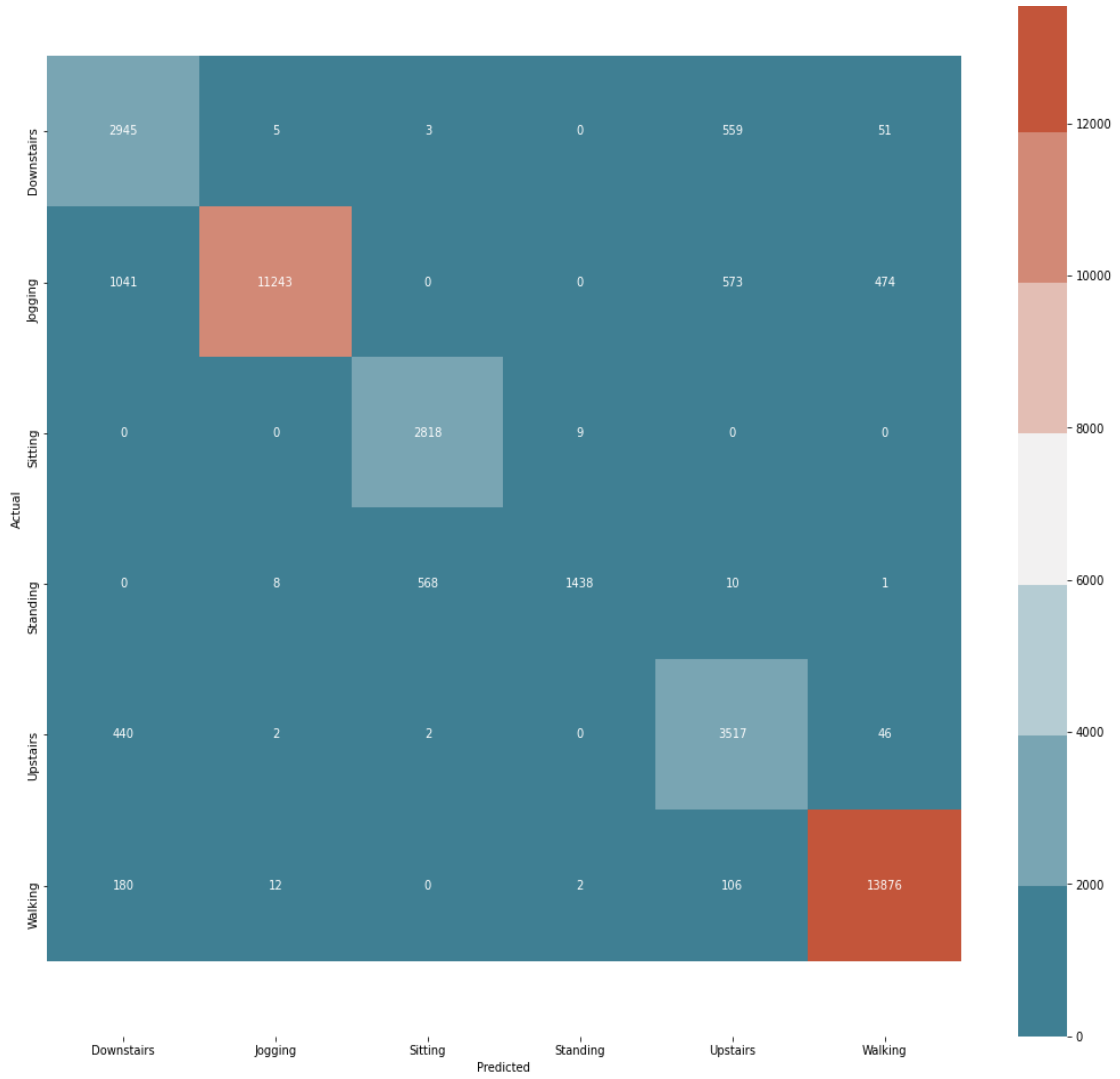
Donc le taux de précision de ce modèle est de 89%.

Pour mieux visualiser les résultats on va construire la matrice de confusion :

In [76]:
```
y_pred = model.predict(X_test)
```

In [77]:
```
from sklearn.metrics import confusion_matrix

def plot_cm(y_true, y_pred, class_names):
  cm = confusion_matrix(y_true, y_pred)
  fig, ax = plt.subplots(figsize=(18, 16))
  ax = sns.heatmap(
      cm,
      annot=True,
      fmt="d",
      cmap=sns.diverging_palette(220, 20, n=7),
      ax=ax
  )

  plt.ylabel('Actual')
  plt.xlabel('Predicted')
  ax.set_xticklabels(class_names)
  ax.set_yticklabels(class_names)
  b, t = plt.ylim() # discover the values for bottom and top
  b += 0.5 # Add 0.5 to the bottom
  t -= 0.5 # Subtract 0.5 from the top
  plt.ylim(b, t) # update the ylim(bottom, top) values
  plt.show() # ta-da!
```

In [78]:
```
plot_cm(
  enc.inverse_transform(y_test),
  enc.inverse_transform(y_pred),
  enc.categories_[0]
)
```



On peut remarquer que notre modèle confond les activités en « Upstairs » et en « Downstairs ». C'est quelque peu attendu car leurs cordonnées et c'est deux activités sont similaires. De plus, selon la distribution des classes on a peu de données pour les catégories « Sitting » ,
« Standing », « Upstairs » et « Downstairs » par rapport à « walking » et « jogging ». Donc le model est bien trainer pour ces deux que les autres catégories.
Pour cela on a essayé de balancer les classes est comparé les résultats :

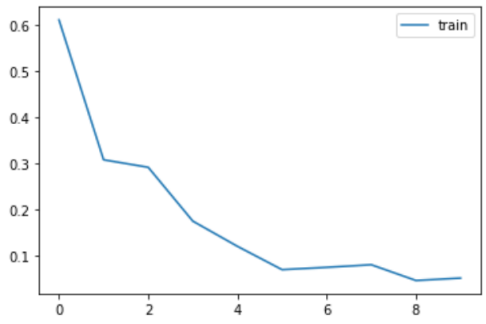
In [10]:
```
df.shape
Walking = df[df['activity']=='Walking'].head(48395).copy()
Jogging = df[df['activity']=='Jogging'].head(48395).copy()
Upstairs = df[df['activity']=='Upstairs'].head(48395).copy()
Downstairs = df[df['activity']=='Downstairs'].head(48395).copy()
Sitting = df[df['activity']=='Sitting'].head(48395).copy()
Standing = df[df['activity']=='Standing'].copy()
balanced_data = pd.DataFrame()
balanced_data = balanced_data.append([Walking, Jogging, Upstairs, Downstairs, Sitting, Standing])
balanced_data.shape
df=balanced_data
```

```
In [11]: sns.countplot(x = 'activity',
                       data = df,
                       order = df.activity.value_counts().index);
```

Voici les nouveaux résultats :

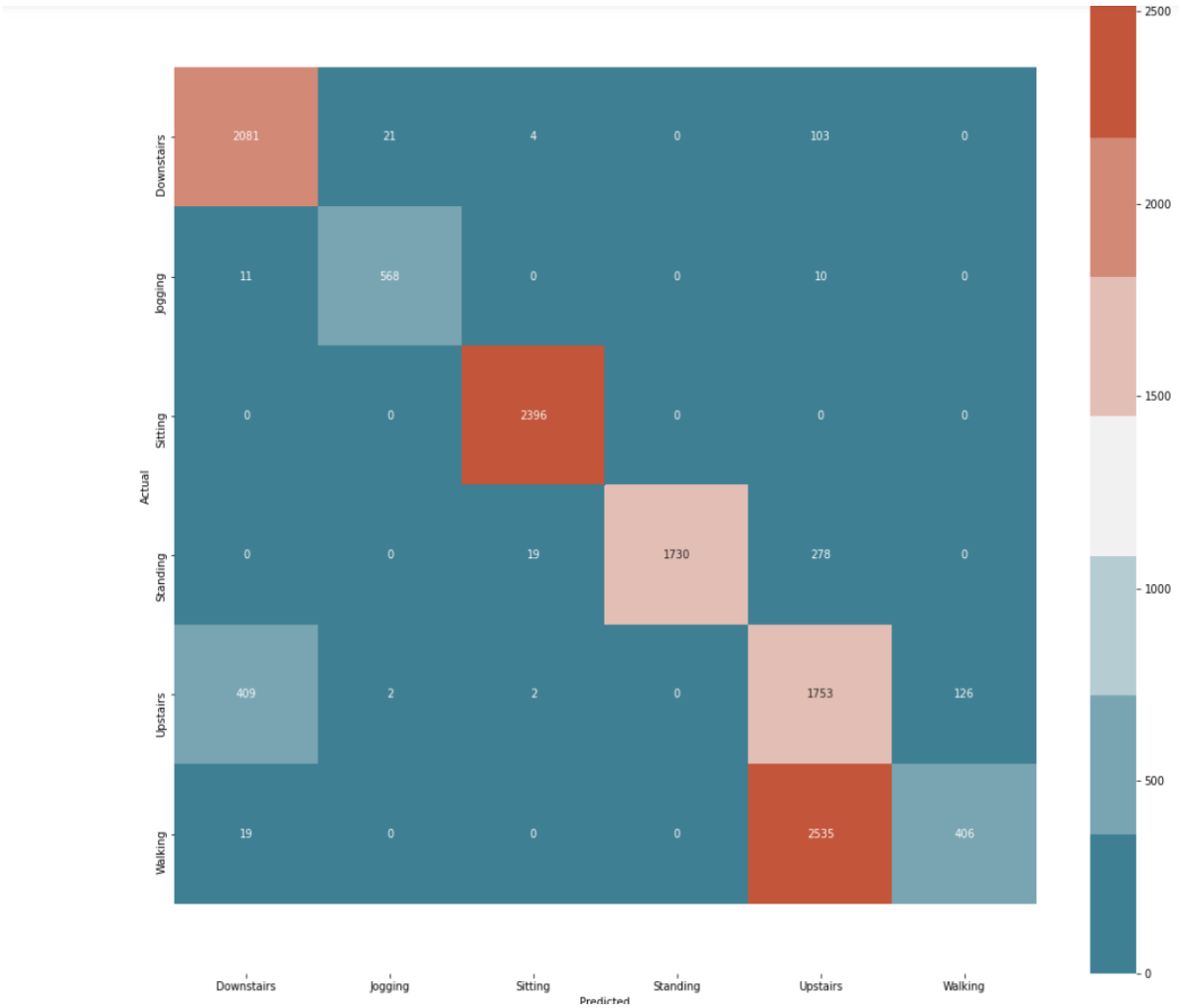
```
In [97]: history=model.fit( X_train, y_train,epochs =10,batch_size=32)

Epoch 1/10
1423/1423 [==============================] - 342s 238ms/step - loss: 0.6103 - acc: 0.7615
Epoch 2/10
1423/1423 [==============================] - 339s 238ms/step - loss: 0.3073 - acc: 0.8770
Epoch 3/10
1423/1423 [==============================] - 340s 239ms/step - loss: 0.2910 - acc: 0.8907
Epoch 4/10
1423/1423 [==============================] - 335s 235ms/step - loss: 0.1743 - acc: 0.9381
Epoch 5/10
1423/1423 [==============================] - 342s 240ms/step - loss: 0.1197 - acc: 0.9578
Epoch 6/10
1423/1423 [==============================] - 344s 241ms/step - loss: 0.0691 - acc: 0.9781
Epoch 7/10
1423/1423 [==============================] - 342s 240ms/step - loss: 0.0742 - acc: 0.9767
Epoch 8/10
1423/1423 [==============================] - 340s 239ms/step - loss: 0.0798 - acc: 0.9738
Epoch 9/10
1423/1423 [==============================] - 341s 240ms/step - loss: 0.0457 - acc: 0.9846
Epoch 10/10
1423/1423 [==============================] - 339s 239ms/step - loss: 0.0508 - acc: 0.9836
```

```
In [98]: plt.plot(history.history['loss'], label='train')
         plt.legend();
```




```
In [99]: model.evaluate(X_test, y_test)

         390/390 [==============================] - 54s 135ms/step - loss: 1.9903 - acc: 0.7163

Out[99]: [1.9902746677398682, 0.7162671089172363]
```

Donc on remarque que le taux de précision c’est diminué à 71%.
Et la nouvelle matrice de confusion :



Donc on peut remarquer que le balancement n'a pas améliorer les résultats et c'est probablement a cause de nombres de données réduit par rapport a la premiers fois .

5. Les problèmes rencontrés dans le TP :

5.1. La parallélisation :

Le temps de construction du model ainsi que le temps d'entraînement est vraiment élevé, pour cela on essaye de faire la parallélisation en utilisant cloud azure est spark . Mais comme on a un compte étudiant on a le droit pour une seule machine virtuelle et on ne peut pas importer le dataset sans un compte premium.

6. Les références :

Apprentissage profond — Wikipédia (wikipedia.org)
Qu'est-ce que le Deep Learning et comment ça marche ? - Saagie
Découvrez les cellules à mémoire interne : les LSTM - Initiez-vous au Deep Learning - OpenClassroomsUnderstanding LSTM Networks -- colah's blog