INTRODUCTION AU COURS

SOMMAIRE

Le cours d'apprentissage profond se propose d'introduire et d'implémenter quelques modèles de réseaux profonds classiques. Les développements informatiques seront réalisés en langage Python, en utilisant, en plus des modules classiques (`numpy`, `matplotlib`...) des bibliothèques dédiées à la création, l'apprentissage et l'utilisation de résdaux profonds. Parmi l'ensemble des bibliothèques disponibles, nous avons choisi de travailler en `TensorFlow` et `Keras`, qui sont détaillés ci-après. D'autres choix sont également possibles, comme le très populaire module `Pytorch`.

1- TENSORFLOW

`TensorFlow` est une bibliothèque open source permettant d'effectuer des calculs numériques en utilisant des graphes de flux de données (version 1.X) ou un mode de programmation plus classique (version 2.X). Cela permet également de déployer le programme sur un ordinateur, sur un serveur ou encore un appareil

mobile avec une seule API. TensorFlow a été initialement développé par les chercheurs et les ingénieurs de l'équipe Google Brain, au sein du département de recherche sur l'intelligence artificielle de Google, dans le but d'être utilisé en apprentissage automatique. Mais le système est tout aussi bien applicable dans une grande variété d'autres domaines.
Nous présentons dans la suite les deux grandes versions de TensorFlow, 1.X et 2.X, et commençons par quelques remarques générales.

1.1- Introduction

1.1.1- Fonctionnement

TensorFlow utilise des tenseurs (figure 1-1) pour représenter toutes les données. Les tenseurs sont des tableaux de données multidimensionnels et de taille dynamique. Cette structure peut donc avoir différents niveaux de complexité, du scalaire à une matrice de dimension n.

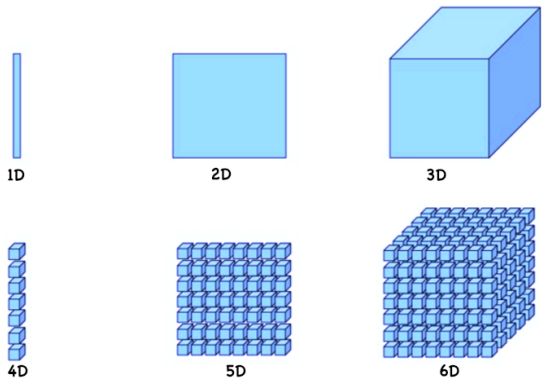

FIGURE 1-1 – Exemples de tenseurs

Les opérations réalisées par TensorFlow découlent directement du ou des modèles utilisés pour l'apprentissage. Or, ces modèles reposent eux même sur des paramètres. Lors de la phase d'entrainement, TensorFlow utilise des variables pour mettre à jour ces paramètres sur lesquels repose le modèle d'apprentissage. Ces variables contiennent des tenseurs, mais contrairement aux tenseurs, les variables sont modifiées tout au long de la phase d'apprentissage.

1.1.2- Avantages et inconvénients

— Paradigme de programmation déclarative : TensorFlow permet de se concentrer sur le problème (quoi), plutôt que sur la solution (comment).
— Portabilité : TensorFlow fonctionne sur CPU, GPU ou TPU. Le modèle créé avec TensorFlow peut être indifféremment déployé sur ordinateur, serveur, appareil mobile et même dans le cloud.
— Flexibilité : TensorFlow ne propose pas que des algorithmes, il propose également un ensemble d'outils permettant d'assembler différentes couches dans les réseaux de neurones. L'utilisateur peut alors créer ses propres algorithmes et les intégrer très facilement.
— Performance : TensorFlow gère lui même les calculs asynchrones et permet donc en théorie de tirer le meilleur parti du matériel disponible.
— API multilangage : TensorFlow est utilisable en Python et en C++ (l'API Python est à l'heure actuelle le plus complet et le plus facile à utiliser, mais l'API C ++ peut offrir certains avantages de performance dans l'exécution du graphe, et prend en charge le déploiement Android).
— Outils incorporés : Tensorboard aide à construire et à visualiser les modèles de flots de données.
— Communauté active : Il suffit de jeter un oeil à la liste des auteurs de TensorFlow pour se rendre compte de l'importance du projet pour Google. Google a rendu cette technologie open source pour accélérer sa croissance.

1.2- Tensorflow 1.X

`TensorFlow 1.x` représente les flux de données sous forme de graphes, où les noeuds représentent des opérations et les arêtes représentent des tenseurs. Ces graphes sont des représentations complètes des calculs qui sont ensuite exécutés sur le CPU, le GPU ou le TPU.

1.2.1- Exemple - classification du problème XOR

Pour prendre en main cette version de `TensorFlow`, on s'intéresse en premier lieu au problème simple de classification XOR à l'aide d'un réseau de neurones "traditionnel" (pas profond). Le code `xor.py` est fourni pour test.

Données et variables
La base d'apprentissage est donc composée de 4 couples $(x_n, y_n)_{1 \leq i \leq 4}, x_n \in \{0,1\}^2, y_n \in \{0,1\}$

$$\mathcal{E}_a = \{(0,0,0),(0,1,0),(1,0,1),(1,1,0)\}$$

Ces couples cont déclarés dans des tableaux `numpy` :

```
X=[[0,0],[0,1],[1,0],[1,1]]
Y = [[0] ,[1] ,[1] ,[0]]
```

`TensorFlow` donne accès aux `placeholder`, des variables symboliques qui représentent les données lors des calculs (apprentissage) du réseau. On définit donc ces conteneurs, qui seront alimentés par la base d'apprentissage lors de la phase d'entraînement, et par des données test dans la phase de génération. Le premier paramètre (de taille) est ainsi mis à `None`, pour pouvoir spécifier des tailles différentes (4 en phase d'entraînement, quelconque pour le reste).

```
x_ = tf.placeholder(tf.float32 , shape=[None,2])
y_ = tf.placeholder(tf.float32 , shape=[None,1])
```

Modèle Le modèle de réseau de neurone est ensuite construit. Ici on s'intéresse à un réseau de neurones à une couche cachée (voir cours prochain).

```
# nombre de neurones caches
hidden_units = 3

# matrice des poids et biais de la premiere couche
b1 = tf.Variable(tf.zeros([hidden_units]))
W1 = tf.Variable(tf.random_uniform([2,hidden_units], -1.0, 1.0))

#  activation non lineaire de la couche cachee
O = tf.nn.sigmoid(tf.matmul(x_,W1)+b1)

# matrice des poids et biais de la sedonde couche
W2 = tf.Variable(tf.random_uniform([hidden_units , 1], -1.0, 1.0))
b2 = tf.Variable(tf.zeros([1]))

# sortie du rÃ©seau
y = tf.nn.sigmoid(tf.matmul(O, W2) + b2)
```

La fonction `matmul()` calcule un produit de tenseurs. `Variable()` est le constructeur de la classe variable, s'initialisant avec un tenseur. La fonction `random_uniform()` retourne un tenseur de la dimension spécifiée, initialisé à l'aide d'une loi uniforme sur l'intervalle spécifié. Le module `nn` contient des fonctions de calcul sur les réseaux de neurones, par exemple des fonctions d'activation prenant en entrée un tenseur et évaluant la transformation non linéaire correspondante(ici `tf.nn.sigmoid()`).

Fonction de perte et algorithme d'optimisation `TensorFlow` fournit des fonctions permettant de manipuler les tenseurs. Ainsi, dans le code précédent, la fonction `reduce_sum()` réduit un tenseur d'une (ou plusieurs) dimension(s), en sommant selon la (ou les) dimension(s) spécifiée(s).

Le module `train` donne accès à de nombreux algorithmes d'optimisation, et ici la fonction `GradientDescentOptimizer` est utilisée.

```
# Fonction de perte quadratique
cost = tf .reduce_sum(tf .square(y_ - y), reduction_indices=[0])
# Optimisation par descente de gradient avec un learning rate de 0.1
train_step = tf.train.GradientDescentOptimizer(0.1).minimize(cost)
```

Session `TensorFlow` A ce stade, les variables ne sont pas encore initialisées. Le graphe sous-jacent est encore symbolique, et il va être instantié avec le démarrage d'une session (par exemple `x_` et `y_` vont recevoir des valeurs).

```
# Initialisation des variables
init = tf.global_variables_initializer()

# Creation d'une session TF pour executer le programme
with tf.Session() as sess:
    sess.run(init)
```

La fonction `Session()` créée une instance de la classe session, et la méthode `run()` pousse le graphe sur le CPU ou le GPU et le remplit avec les variables initialisées (variable `init`).

Entraînement et évaluation On réalise des itérations de la descente de gradient. L'appel à `sess.run()` enchaîne les opérations précédemment définies.

```
num_iter = 5000

correct_prediction = abs(y_ - y) < 0.5
cast = tf.cast(correct_prediction , "float")
precision = tf .reduce_mean(cast)

# Creation d'une session TF pour executer le programme
with tf.Session() as sess:
        sess.run(init)
        #nombre d'iterations
        for i in range(num_iter):
                sess.run(train_step,feed_dict={x_: X, y_: Y})
                if i % 100 == 0:
                        loss, acc = sess.run([cost, precision], feed_dict={x_: X, y_: Y
                            })
                        print("Iteration " + str(i) + ", Cout  = ",loss, ", PrÃ©cision
                            = " + "{:.5f}".format(acc))
```

Visualisation des résultats Pour visualiser les résultats (figure 1-2), on utilise `matplotlib`.

```
plt.figure()
c1 = plt.scatter([1,0], [0,1], marker='s', color='red', s=100)
c0 = plt.scatter([1,0], [1,0], marker='o', color='gray', s=100)
# Generation de  points dand [-1,2]x[-1,2]
DATA_x = (np.random.rand(10**6,2)*3)-1
DATA_y = sess.run(y,feed_dict={x_: DATA_x})
# Predictions
ind = np.where(np.logical_and(0.49 < DATA_y, DATA_y< 0.51))[0]
DATA_ind = DATA_x[ind]
# Surfaces de separation
ss = plt.scatter(DATA_ind[:,0], DATA_ind[:,1], marker='_', color='blue', s=2)


plt.legend((c1, c0, ss), ('Classe 1', 'Classe 0', 'Surfaces de sÃ©paration'),
    scatterpoints=1)
plt.xlabel('x1')
plt.ylabel('x2')
plt.axis([-1,2,-1,2])
plt.show()
```

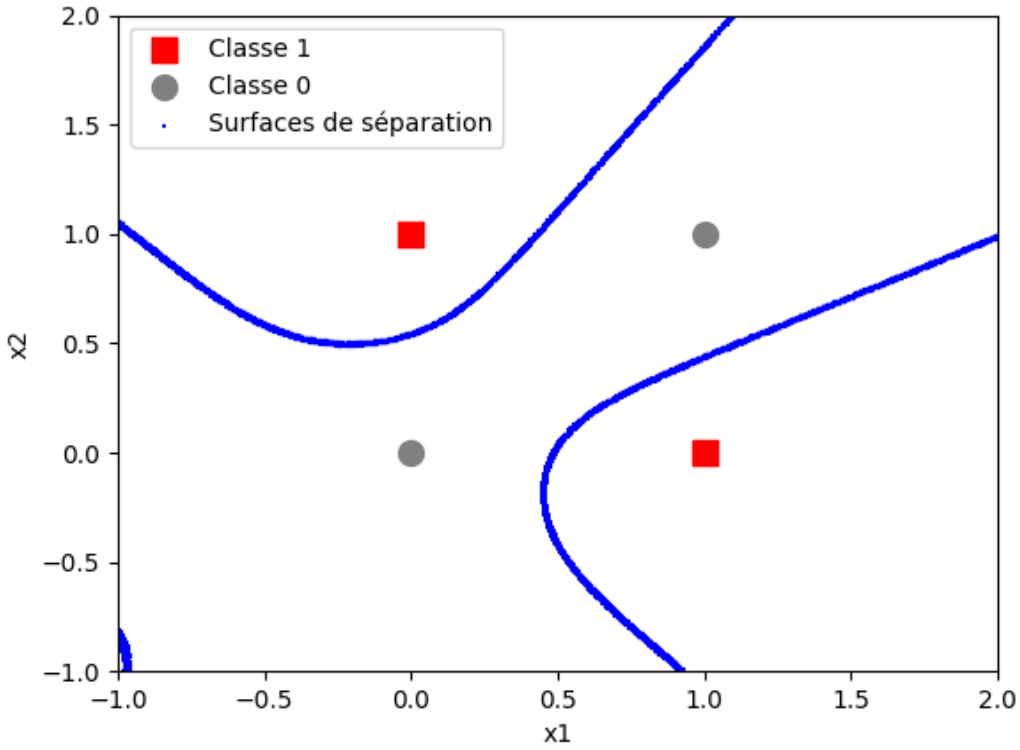

FIGURE 1-2 – Résultat de la classification du réseau sur les données XOR

1.2.2- Mode "Eager execution"

Introduit dans les dernières versions de `TensorFlow`, et amené à prendre une part prépondérante dans la version 2.0 (voir paragraphe 1.3.1-), le mode "Eager Execution" permet d'utiliser `TensorFlow` via des commandes orientées objet.
L'utilisation classique de `TensorFlow`, dans les premières versions, étaient orientée graphe. L'apparition d'un mode d'exécution direct présente de nombreux avantages : facilité de debuggage, construction dynamique du code, ajout facile d'opérations et de code..
L'utilisation est simple, il suffit d'ajouter en préambule, après les imports, la commande `tf.enable_eager_execution()`. Les commandes s'enchaînent ensuite, classiquement, sans faire appel à une session ni à l'exécution d'un graphe.
D'un point de vue performances, et même si le mode d'exécution graphe était optimisé, les concepteurs affirment qu'il n'y a quasiment pas d'impact lors de l'entraînement et optimisation de gros modèles. Cependant il est toujours possible de passer du mode eager au mode graphe à l'aide du décorateur tensorflow.function.

1.2.3- Tensorboard

Introduction TensorBoard est une interface de monitoring graphique de `TensorFlow`. Il permet de suivre le flot des données dans le graphe qui représente les calculs effectués sur le réseau de neurones. L'enchaînement des opérations dans un grand réseau peut en effet vite s'avérer complexe. TensorBoard permet de visualiser le graphe créé, ainsi que de contrôler l'évolution des opérations effectuées lors de la phase d'apprentissage (évolution du taux de prédiction, activités des neurones, etc.).
Pour utiliser TensorBoard, il convient de spécifier dans le code quelles sont les opérations dont on souhaite résumer l'activité. Il reste ensuite à créer un `Summarizer` qui agrège toutes les informations calculées. Ces

informations sont écrites dans des fichiers log. TensorBoard exploite ces logs pour construire un graphe, qui peut être visualisé par l'appel de `tensorboard - -logdir="chemin_des_logs"`. Le visualisateur est un navigateur internet et l'adresse est `http://0.0.0.0:6006/`.

Exemple Reprenons l'exemple du XOR. Le fichier `xor_tensorboard.py` donne un aperçu de quelques fonctionnalités du TensorBoard.
Les variables à suivre sont encapsulées dans un scope à l'aide de la méthode `name_scope`. On peut suivre un ensemble de variables (code .1) ou plusieurs variables indépendamment (code .2)

```
with tf.name_scope("Model") as scope:
        # matrice des poids et biais de la premiÃ¨re couche
        b1 = tf . Variable ( tf . zeros ([hidden_units]))
        W1= tf.Variable(tf.random_uniform([2,hidden_units], -1.0, 1.0))
        #activation non linÃ©aire de la couche cachÃ©e
        O=tf.nn.sigmoid(tf.matmul(x_,W1)+b1)
        # matrice des poids et biais de la seconde couche
        W2= tf.Variable(tf.random_uniform([hidden_units,1], -1.0, 1.0))
        b2 = tf . Variable ( tf . zeros ([1]))
        #sortie du rÃ©seau
        y = tf.nn.sigmoid(tf.matmul(O, W2) + b2)
```

Listing .1 – scope sur un groupe de variables

```
with tf.name_scope("cost") as scope:
        # Fonction de perte quadratique
        cost = tf .reduce_sum(tf .square(y_ - y))

with tf.name_scope("optim") as scope:
        # Optimisation par descente de gradient, learning rate=0.1
        train_step = tf.train.GradientDescentOptimizer(0.1).minimize(cost)

with tf.name_scope('Precision'):
        correct_prediction = abs(y_ - y) < 0.5
        cast = tf.cast(correct_prediction , "float")
        precision = tf .reduce_mean(cast)
```

Listing .2 – scope sur une variable

Une fois les variables étiquetées, on précise que l'on veut les intégrer à un objet `summary` qui tracera les valeurs de ces variables et on rassemble l'ensemble des variables dans cet objet.

```
tf.summary.scalar("cost", cost)
tf.summary.scalar("precision", precision)

# Suivi des variables du modÃ¨le
for var in tf.trainable_variables():
    tf.summary.histogram(var.name, var)
merged_summary_op = tf.summary.merge_all()
```

A l'ouverture de la session, il reste à écrire les valeurs de ces variables dans le résumé à l'aide des appels à

```
summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_graph())
```

et à

```
_, c, summary = sess.run([train_step, cost, merged_summary_op],feed_dict={x_: X, y_: Y})
summary_writer.add_summary(summary, i)
```

Les résultats graphiques sont présentés sur les figures 1-3 à 1-6

Debugger avec Tensorboard TensorFlow Debugger est une interface web interactive (liée à tfdbg, le debugger en ligne de commande introduit dans les dernières versions de `TensorFlow`) , facilement intégrée à Tensorboard, pour contrôler l'exécution des modèles `TensorFlow`, mettre des checkpoints, naviguer dans les noeuds....

FIGURE 1-3 – Evolution du coût et de la précision

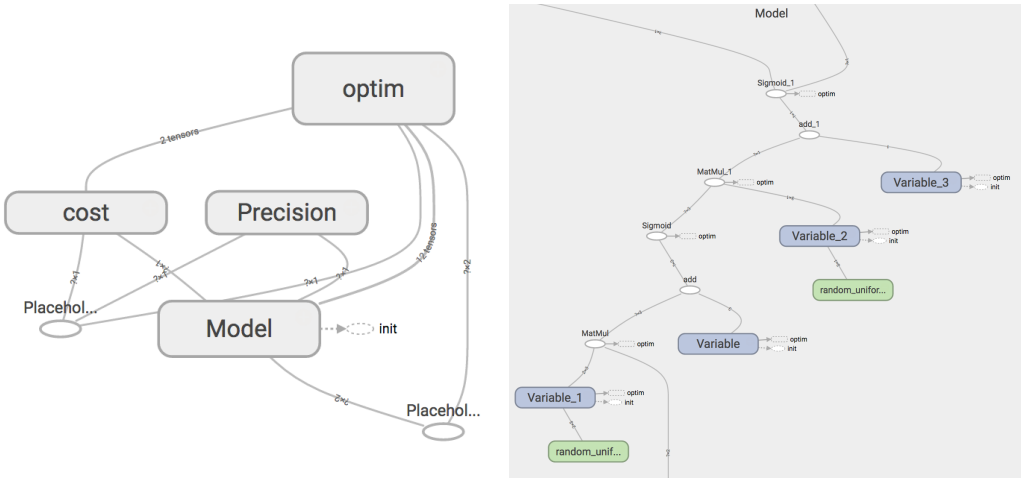

FIGURE 1-4 – Graphe et graphe du modèle

L'utilisation du debuggeur s'effectue en ajoutant sa référence dans l'appel à TensorBoard lors de son lancement en ligne de commande :

```
tensorboard --logdir /tmp/logdir --debugger_port 7000
```

1.3- TensorFlow 2.X

1.3.1- Mode Eager

Le mode "Eager" permet de s'affranchir des notions de graphe et de session, et d'utiliser des tenseurs de manière très naturelle et classique.

```python
import tensorflow as tf

a = tf.random.normal(shape=(4,4))
b = tf.random.normal(shape=(4,4))
c = tf.ones(shape=(1))

result = c + a * b
print(result)
```

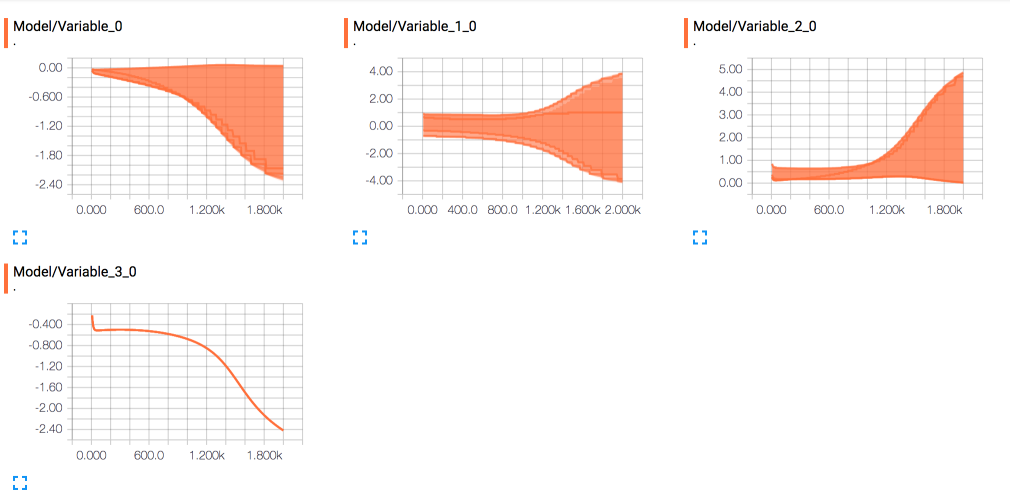

FIGURE 1-5 – Distributions des variables suivies

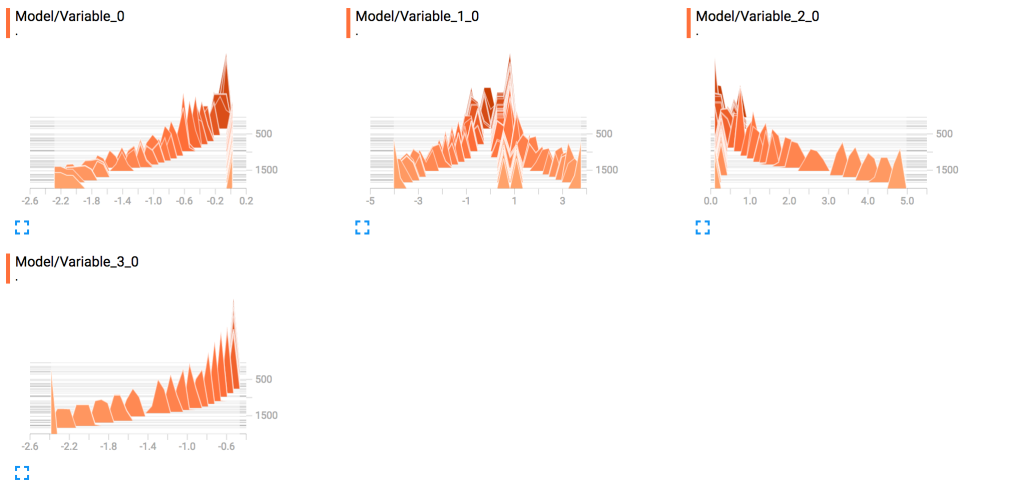

FIGURE 1-6 – Histogrammes des variables suivies

La manipulation de variables qui changent lors de l'entraînement (les poids et biais d'un réseau de neurones par exemple), devient elle aussi très facile.

```
import tensorflow as tf

x = tf.random.normal(shape=(1000, 10))

w = tf.Variable(tf.random.normal(shape=(10, 2)))
b = tf.Variable(tf.random.normal(shape=(2,)))

y_pred = tf.matmul(x , w) + b

print(y_pred)
```

et l'apprentissage d'un modèle se lit également facilement (ici on utilise `Keras`, qui sera abordé ultérieurement)

```
import tensorflow as tf
import numpy as np

x_train = np.random.random((1000, 20))
y_train = np.random.randint(2, size=(1000, 1))
x_test = np.random.random((100, 20))
y_test = np.random.randint(2, size=(100, 1))

model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(64, activation='relu', input_dim=20),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(1, activation='softmax')
        ])


model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
          epochs=20,batch_size=128)
score = model.evaluate(x_test, y_test, batch_size=128)
```

Une documentation complète sur le mode Eager est disponible ici
Le notebook associé à ce cours détaille d'autres aspects de `Tensorflow` 2.0 (GradientTape, accélération du code par le retour à un graphe de flux de données en utilisant le décorateur tensorflow.function...)

1.3.2- Sauvegarde et utilisation d'un modèle

La sauvegarde d'un modèle sous `Tensorflow` 2.X est très simple :
`model.save_weights('nom.hd5')`
Pour restaurer les poids d'un réseau, il faut, comme sous les versions 1.X, déclarer l'architecture d'un réseau, puis appeler la fonction
`model.load_weights('nom.hd5')`

1.3.3- Pour utiliser `Tensorflow` 2.0

Pour utiliser la version 2, deux options :

- soit installer la version 2
- soit rester en version 1.13 ou 1.14, et utiliser les commandes `import tensorflow.compat.v2 as tf` et `tf.enable_v2_behavior()`

2- KERAS

Il est possible d'utiliser `Tensorflow` avec des API haut niveau, permettant un prototypage rapide d'applications de deep learning. Parmi ces API, `Keras` a rapidement connu un succès dès son introduction et en 2017 a été intégrée au coeur de Tensorflow en tant que `tf.keras`. Bien que `tf.keras` et `Keras` aient des codes séparés, ils sont intimement couplés. La documentation et le guide du programmeur de `Tensorflow` 1.9 suggèrent fortement que `tf.keras` est l'API à utiliser pour créer des réseaux de neurones avec `Tensorflow`.
`Keras` attaque également d'autres bibliothèques comme Theano, Microsoft Cognitive Toolkit ou PlaidML.

Le notebook associé à ce cours donne plusieurs exemples d'utilisation de `Keras`, que nous reprenons pour certains ici.

2.1- Prise en main de Keras

La principale structure de données dans `Keras` est le modèle, qui permet de définir complètement le graphe représentant le réseau. `Keras` propose deux manières pour construire un modèle :

1. l'approche séquentielle, pour la création de modèles simples
2. l'approche fonctionnelle, pour les modèles complexes.

2.1.1- Modèle séquentiel

Le code suivant montre les imports nécessaires à l'utilisation d'un modèle séquentiel. Le module `keras.layers` contient toutes les couches classiques d'un réseau de neurones (convolution, pooling, diverses activations, normalisation, Dropout, LSTM...).

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Conv2D, MaxPooling2D
```

La génération du modèle se fait alors couche à couche, et pour générer un modèle simple (voir cours correspondant) comportant :

- une couche d'entrée de convolution à 32 filtres de taille 3×3, avec activation ReLU, acceptant des images en entrée de taille $128\times128\times3$.
- une couche de pooling max sur un voisinage 4×4
- une couche complètement connectée à 256 sorties
- une couche de classification, avec un classifieur softmax

on propose le code suivant :

```
model = Sequential()
model.add(Conv2D(32,(3,3),activation='relu',input_shape=(128,128,3)))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Dense(256))
model.add(Activation('softmax'))
```

Il s'agit ensuite de définir la fonction de perte et l'algorithme d'optimisation, et par exemple pour minimiser l'entropie croisée par l'algorithme rmsprop, on écrit simplement

```
model.compile(loss='binary_crossentropy',optimizer='rmsprop')
```

En supposant que les données d'apprentissage (resp de validation, de test) ont été stockées dans des tableaux `X` et `Y` (resp. `vXal` et `yVal`, `xTest` et `yTest`), l'entraînement, le test et la sauvegarde du modèle s'effectuent par un appel à

```
#Entrainement
model.fit(X,Y,batch_size=32, epochs=10,validation_data=(xVal, yVal))
#Evaluation
score = model.evaluate(xTest,yTest,batch_size=32)
#Sauvegarde
model.save_weights("modele.h5")
```

2.1.2- Approche fonctionnelle

Le modèle séquentiel n'est envisageable que pour des modèles simples. Dans les modèles actuels, l'architecture des réseaux est plutôt organisée autour d'un "mini-réseau" (module Inception, module Fire dans SqueezeNet,...) qui se répète de nombreuses fois. L'approche fonctionnelle permet d'appeler des modèles simples de la même manière qu'une couche est appelée dans l'approche séquentielle.
Le code ci-après décrit les principales étapes de définition d'un réseau par l'approche fonctionnelle. Après avoir importé `Model`, il s'agit de définir les entrées, en important `Input`, puis en définissant (ici) un tenseur de taille $1\times128\times128$. On applique alors (ici) deux couches de convolution (en spécifiant la variable à laquelle appliquer la couche), puis une agrégation max et une couche `Flatten`. Le modèle est enfin créé.

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input

#DÃ©finition des entrees
entree = Input(shape=(1, 128, 128))

#Application d'une couche de convolution
x = Conv2D(32, (3, 3))(entree)
# Puis une convolution
x = Conv2D(64, (3, 3))(x)
# puis un max pooling
x = MaxPooling2D((4, 4))(x)
# puis une couche Flatten pour obtenir la sortie
out = Flatten()(x)

# Modele
my_model = Model(entree, out)
```

La définition de la fonction de perte, de l'algorithme d'optimisation et de l'entraînement sont identiques au cas du modèle séquentiel.

2.1.3- Visualisation du réseau

Il est possible, à la manière de TensorBoard, de visualiser le graphe du réseau produit avec `Keras`. Pour ce faire, après avoir installé `graphviz, pydot-ng`, un import de `keras.utils.plot_model` et un appel à

```
plot_model(model,to_file='model.png',show_shapes=True,show_layer_names=True)
```

permet d'obtenir un graphe comme celui présenté en figure 2-7

2.2- Tensorflow, mode eager et Keras

La combinaison de `Keras` executé en mode eager et de `Keras` offre la possibilité de réaliser rapidement et facilement des algorithmes d'apprentissage.Le notebook adossé à ce cours en donne un exemple.

3- DONNÉES

3.1- Données simulées

Dans certains TP, il sera donné ou demandé de synthétiser des données simulées, pour tester de manière simple les algorithmes développés. Il pourra par exemple s'agir de données linéairement séparables, ou au contraire non linéairement séparables. D'autres types de données, spécifiques (données séquentielles) seront également utilisées.

3.2- MNIST

La base de données MNIST (Mixed National Institute of Standards and Technology), est une base de données de chiffres manuscrits. C'est une base de données standard pour le test de nouvbeaux algorithmes de reconnaissance de ces chiffres. Elle est composée de 60000 images d'apprentissage et 10000 images de test,. Les images en noir et blanc, normalisées centrées de 28 pixels de côté (figure 3-8)

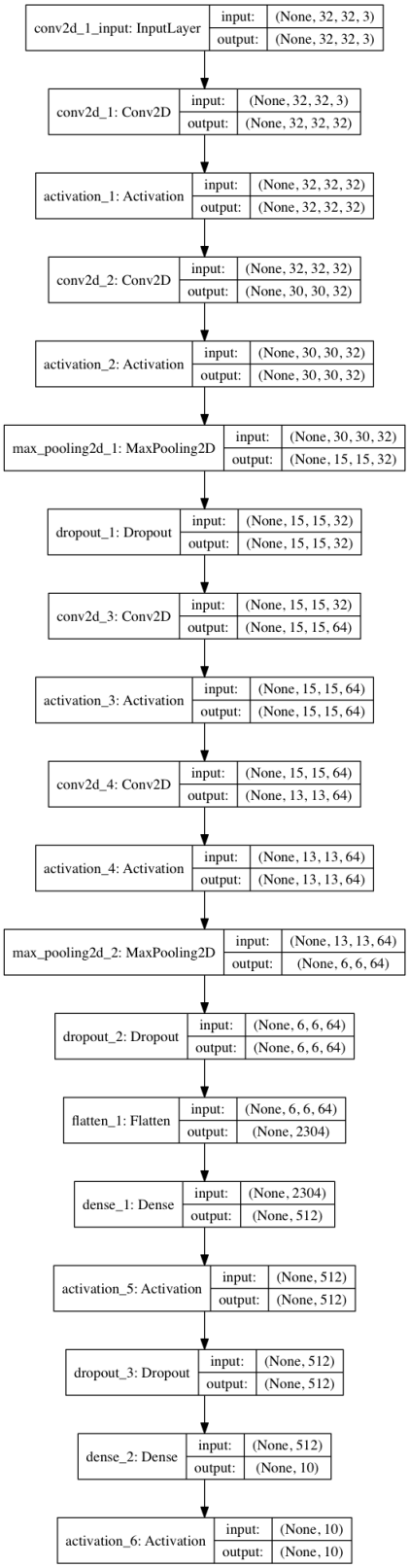

FIGURE 2-7 – Graphe du réseau

FIGURE 3-8 – Exemple d'images tirées de la base MNIST

4- ENVIRONNEMENT DE DÉVELOPPEMENT DANS LES LOCAUX DE L'ISIMA

Les différents TP se feront avec le langage de programmation Python. Afin d'être maître de l'installation des paquets Python, nous allons utiliser des environnements virtuels sous VirtualEnv. De plus, afin d'avoir un code commenté et expliqué pour comprendre les différents concepts nous allons utiliser des notebooks sous Jupyter.

4.1- Machine TURING / répertoires `$HOME` et `$HOME/shared`

Les données de vos sessions (téléchargements, travaux, ...) sont stockées dans votre répertoire `$HOME`. Les données que vous avez dessus sont persistantes, mais ne sont en aucun sauvegardées et il n'existe aucune garantie qu'une opération de maintenance ne détruise pas les données qui s'y trouvent.

Vous avez donc à votre disposition un répertoire dans votre `$HOME` nommé `$HOME/shared`. Il s'agit d'un espace de stockage qui vous est personnel, partagé sur le réseau et qui le rend accessible depuis toutes les machines pédagogique Linux et Windows. Le contenu est sauvegardé tous les soirs.

La bonne pratique à observer est de travailler dans votre `$HOME`, puis à la fin de votre séance de sauvegarder ce dont vous avez besoin dans votre répertoire `$HOME/shared` pour être sur de bien retrouver vos travaux dans le futur.
Cependant, il est fortement **déconseillé de travailler directement dans ce répertoire**. Par exemple de sauvegarder un projet de l'IDE de votre choix dans votre répertoire `$HOME/shared`, de lancer des compilations dans ce répertoire ou d'y créer un environnement virtuel Python ...

Travailler directement dans le répertoire `$HOME/shared` va causer des ralentissements sur les programmes que vous utilisez. En effet un programme qui fait de nombreuses opérations de lecture et/ou d'écriture dans ce répertoire sera ralenti. Ce répertoire étant sur le réseau les opérations sont lentes et ralentissent le processus.

Pour résumer :

1. Copiez vos travaux précédemment sauvegardés depuis `$HOME/shared` dans votre `$HOME` si vous en avez besoin pour travailler ;
2. Travaillez dans votre `$HOME` ;
3. En fin de séance copiez vos travaux à nouveau sur `$HOME/shared`.

4.2- Utilisation de Virtualenv

Virtualenv est un outil pour créer des environnements virtuels Python isolés. Virtualenv crée un dossier qui contient tous les exécutables nécessaires pour utiliser les paquets qu'un projet Python pourrait nécessiter. Un guide de survie est proposé sur la page de documentation utilisateur du service informatique de l'ISIMA. Pour configurer Virtualenv voici les étapes à réaliser :

— Créer un répertoire dans son répertoire utilisateur (sous `$HOME`) pour recevoir l'environnement virtuel (`mkdir`) ;
— Créer l'environnement virtuel en spécifiant l'interpréteur Python de votre choix (`virtualenv`) ;
— Activer l'environnement virtuel (`source`) ;
— Installer des paquets Python (`pip`) .

Quand vous avez terminé de travailler dans l'environnement, il faut le désactiver : `deactivate`.
Voici les lignes à exécuter dans le terminal pour créer un environnement virtuel :

```
mkdir DLVirtualEnv
virtualenv -p python3 DLVirtualEnv
```

Ensuite, pour lancer un environnement virtuel il suffit d'exécuter

```
source DLVirtualEnv/bin/activate
```

Pour vérifier le bon fonctionnement de l'environnement virtuel, on peut remarquer que le prompt est devenu : `(DLVirtualEnv) [login@turing myTP]`. Pour désactiver l'environnement virtuel, il faut écrire dans le terminal la commande suivante :

```
deactivate
```

Dans cet environnent virtuel, il n'y aucun (ou peu) de paquets d'installés, on va pouvoir personnaliser cet environnement avec les paquets avec les versions adaptées. On peut installer l'ensemble des paquets par la commande suivante :

```
pip install -r requirements.txt
pip install -r requirements2.txt
```

Le fichier `requirements.txt` contient une liste simple de tous les paquets et leurs versions respectives qui seront installés par la commande `pip`. Ce fichier texte contient les lignes suivantes :

```
tensorflow==2.0
matplotlib
pandas
jupyter
ipykernel
ipdb
```

Le fichier `requirements2.txt` est le suivant :

```
tornado
jupyter_tensorboard
```

Lorsqu'il faudra un nouveau paquet pour un TP, il suffira de réaliser un `pip` dans votre environnement virtuel.

4.3- Utilisation de Jupyter

Jupyter est une application web utilisée pour programmer dans plus de 40 langages de programmation, dont Python. Jupyter permet de réaliser des notebooks, c'est-à-dire des programmes contenant à la fois du texte en markdown et du code en Python dans notre cas.
Dans le paragraphe précédent, le paquet Jupyter dans l'environnement virtuel a déjà été installé. Pour l'utiliser, il faut rajouter le nouveau noyau (environnement virtuel) dans l'outil Jupyter, par la commande :

```
python -m ipykernel install --user
       --name DLVirtualEnv
       --display-name "Kernel Python ZZ3 F2F4"
```

Pour travailler, il suffit de lancer Jupyter.

```
jupyter notebook
```

puis de sélectionner le nouveau noyau dans le menu.