Version en ligne
OFFERTE !
pendant 1 an

+ QUIZ

TensorFlow et Keras

L'intelligence artificielle appliquée à la robotique humanoïde

En téléchargement

extraits de code

informatique technique

Collection EPSILON

eni

Henri LAUDE

Table des matières

Les éléments à télécharger sont disponibles à l'adresse suivante :
http://www.editions-eni.fr
Saisissez la référence de l'ouvrage **EPTENS** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

Chapitre 1
Introduction

1. L'esprit de l'ouvrage 11
2. L'Intelligence Artificielle dans cet ouvrage 13
3. Les promesses de TensorFlow 15
4. Zoom sur Keras. 16
5. Comment l'esprit vient aux machines 18
   - 5.1 Des origines du robot humanoïde moderne et de la nécessité de le doter d'une certaine intelligence 18
   - 5.2 De l'évolution des robots humanoïdes et de leurs besoins croissants en intelligence 20
   - 5.3 Les fondements du traitement algorithmique de l'intelligence artificielle 26
   - 5.4 Les probabilités modernes et une initiation rapide et un peu brutale au point de vue bayésien 27
     - 5.4.1 Le point de vue opposé à l'optique bayésienne : l'approche fréquentiste 27
     - 5.4.2 De Bayes aux réseaux bayésiens, en passant par la mesure 28
   - 5.5 L'apprentissage statistique (machine learning) en bref 35
     - 5.5.1 Apprentissage supervisé. 35
     - 5.5.2 Apprentissage non supervisé 36
     - 5.5.3 Apprentissage par renforcement 37
   - 5.6 Les formes d'intelligence 38

5.7 Apprendre de la bêtise 39
6. En guise de conclusion… de l'introduction ! 43
7. Bibliographie du chapitre 44

Chapitre 2
Introduction à TensorFlow et Keras

1. Présentation générale de TensorFlow 45
1.1 Le produit 45
1.2 Les réseaux neuronaux en (très) bref. 46
2. Préparation de l'environnement de travail TensorFlow. 50
2.1 Vérification de l'installation de TensorFlow 50
2.2 Données à disposition pour s'entraîner. 53
2.3 R ou Python … ou un autre langage ? 56

Chapitre 3
Code et mathématiques

1. Présentation du chapitre 59
2. En route vers les tenseurs. 61
2.1 Autour des tenseurs 61
2.2 Contexte mathématique du chapitre 63
3. Les vecteurs. 65
3.1 Approche intuitive des vecteurs 65
3.2 Normes d'un vecteur 68
3.3 Distance issue d'une norme. 69
3.4 Indices et convention d'Einstein 69
3.5 Produit scalaire 70
3.6 Création d'un produit scalaire à partir d'une norme 73
4. Les matrices. 74
5. Les quaternions. 77

6. Les tenseurs 80
6.1 Exemples de formes d'un tenseur 80
6.2 Forme linéaire 82
6.3 Application multilinéaire 84
6.4 Produit tensoriel de vecteurs ou matrices 84
7. Différentiation 87
7.1 Introduction 88
7.2 Généralisation et introduction à la différentiabilité 89
7.3 Quelques cas importants de dérivées totales 91
7.4 Analogie calculatoire 95
7.5 Différentielle 96
7.6 Différentiation automatique (AD) 98
7.7 Zoom sur la notion de quantité négligeable 100
8. Gradient 102
8.1 Dérivée suivant une direction et introduction des dérivées partielles 103
8.2 Définition du Gradient 105
8.3 Considérations pratiques autour du Gradient 106
9. Tenseur métrique 108
10. Optimisation 112
11. Boîte à outils complémentaire 114
11.1 Filtre de Kalman 114
11.2 One hot encoding 115
11.3 Réseaux de neurones "vanilla" 116
12. Bibliographie du chapitre 119

Chapitre 4
Hello world

1. L'exemple de référence 121
   1.1 Présentation du dataset 121
   1.2 Installation de modules complémentaires 123
2. Notre premier réseau de neurones 125
   2.1 Construction "pas à pas" du réseau avec Keras 125
   2.2 En entrée du modèle 126
   2.3 Mise en place de notre première couche d'apprentissage 126
   2.4 Un mécanisme pour éviter l'overfitting : le dropout 130
   2.5 Définir la couche sortie du modèle - couche softmax 131
   2.6 Le code correspondant 132
   2.7 Création d'un modèle instanciable, backpropagation 135
   2.8 Instanciation du modèle 137
   2.9 Évaluation du modèle 140
   2.10 Utilisation du modèle instancié 141
   2.11 Observation directe des résultats 142
3. Conclusion 144
4. Bibliographie du chapitre 145

Chapitre 5
Amélioration matérielle

1. Amélioration des performances via l'utilisation de CUDA 147
   1.1 CUDA et TensorFlow 2 avec GPU 148
   1.2 Exécution du code en utilisant une GPU 152
   1.3 Conclusion 157
2. Encore un peu de tuning 158
   2.1 Sauvegarder des modèles intermédiaires 158
   2.2 Sauver des modèles en JSON ou YAML 160
   2.3 Considérations techniques sur la configuration matérielle . . . 162
3. Bibliographie du chapitre 163

Chapitre 6
Améliorations du modèle

1. Convolution et pooling 165
   1.1 Convolution 165
   1.2 Pooling 173
2. Batch normalization 176
3. Data augmentation 178
4. Bibliographie du chapitre 183

Chapitre 7
Introduction à la robotique anthropomorphiste

1. Ce qu'il nous faut appréhender de la robotique 185
2. Mais qui sont ces robots ? 188
   2.1 Le contexte 188
   2.2 Problèmes clés étudiés depuis les années 1970 189
      2.2.1 Optimal control of motion planning 189
      2.2.2 Model based control - considérations préliminaires . . 194
      2.2.3 Asservissement 204
      2.2.4 Zoom sur les techniques d'asservissement 205
   2.3 Model Based Control (et Model learning) 210
   2.4 Planning and trajectory 211
3. Notions basiques : DOF, Actuator, Effector, DOM 222
   3.1 Le vocabulaire 222
   3.2 La cinématique 223
   3.3 Cinématique du pendule inversé 224
   3.4 L'agencement de modèles différents 225
4. Interfaces avec le robot 228
   4.1 Vers une interface directe avec le robot 228
   4.2 Interfaces sémantiques 229
5. Des données pour s'entraîner 231

6. Bibliographie du chapitre 233

Chapitre 8
Simuler les attributs de l'intelligence

1. Identifier visuellement les objets 235
1.1 Le problème 235
1.2 Éléments de l'état de l'art 236
1.2.1 Les images labellisées 236
1.2.2 Les réseaux neuronaux phares de la reconnaissance d'objets dans des images 239
1.2.3 Un exemple d'implémentation 240
2. Transfert learning 242
3. RNN et LSTM, quand le temps et les sentiments s'en mêlent 244
3.1 Le problème 244
3.2 La solution 245
3.2.1 Les RNN classiques 245
3.2.2 Les RNN à mémoire, dont les LSTM 248
3.3 Une implémentation de LSTM : génération de suites cohérentes de mots 250
4. Passer du son au mot : speech to text 256
4.1 Le problème posé 256
4.2 Une implémentation 257
5. Passer du mot au son : text to speech 259
5.1 Un problème moins simple qu'il n'y paraît 259
5.2 Une implémentation 259
6. Traduction automatique 260
7. Manipuler le langage 262
7.1 Des problèmes courants 262
7.2 Implémentation 262
7.2.1 NLTK (Natural Language Toolkit) et ses compagnons 262

7.2.2 Travailler un texte : tokenization et compagnons. . . . 262
7.3 Zoom sur la notion d'embedding 271
7.4 Particularité d'un modèle simple de classification de textes . . 281
8. Manipuler le son 283
9. Simuler la création et/ou jouer 289
9.1 GAN, un modèle qui donne l'initiative à l'IA 289
9.2 Sec2sec : pour générer des séquences. 293
9.3 VAE - encoder et décoder pour générer des images 293
10. Bibliographie du chapitre 296

Chapitre 9
Confrontation aux réalités opérationnelles

1. Se doter de divers environnements de travail 299
1.1 Le problème. 299
1.2 Les pistes de solution 300
1.3 En pratique 300
1.3.1 Éléments de configuration de base 300
1.3.2 Docker et Kaggle 302
1.4 Mise en exploitation réelle et/ou exposition d'un prototype opérationnel 303
1.4.1 Exposition de services ou d'une interface web (Flask, Django, plumber, TensorFlow.js) 303
1.4.2 Intégration et déploiement continus 307
1.4.3 TensorFlow Lite 308
2. Répartition des traitements 309
2.1 Le problème. 309
2.2 Éléments de solution 309
2.2.1 Préparation des données 309
2.2.2 Création des modèles 311
2.2.3 Exploitation des modèles 311
2.2.4 Travailler sur plusieurs GPU 311

2.2.5 Se préparer au HPC (High Performance Computing) . 312
3. Le cas particulier du robot humanoïde autonome et des systèmes embarqués légers 313
4. Bibliographie du chapitre 314

Conclusion

1. En guise de conclusion 315
2. Remerciements 316

Annexes

1. Introduction aux annexes 317
2. Petits savoir-faire TensorFlow utiles 318
2.1 Transformer un tenseur Numpy en tenseur TensorFlow 318
2.2 Les fonctions d'activation de TF 318
2.3 Création d'une classe qui produit un modèle Keras 319
2.4 Création d'un modèle avec plusieurs branches 319
2.5 Batch normalization 321
2.6 Choisir un optimiseur 322
2.7 La régularisation : diminuer le risque d'overfitting 322
2.8 Manipulations de tenseurs 323
2.8.1 Exemple de réduction sur un axe 323
2.8.2 Map : application d'une fonction sur les composants d'un tenseur 325
2.9 Matrice adjointe d'une matrice (à coefficients complexes) . . . 326
2.10 Transformation des lignes d'une matrice en termes diagonaux 326
2.11 Notation d'Einstein 327
2.11.1 Produit scalaire de deux vecteurs 327
2.11.2 Produit dyadique de deux vecteurs 328

2.11.3 Produit d'Hadamard de deux matrices 328
2.11.4 Produit matriciel. 329
2.11.5 Application d'une matrice sur un vecteur 330
2.11.6 Application d'un vecteur transposé sur une matrice . . 330
2.11.7 Scalaire par scalaire, vecteur, matrice. 331
2.11.8 Somme des éléments d'un tenseur 331
2.11.9 Trace d'une matrice 332
2.11.10 Produit de Kronecker de deux matrices 332
2.11.11 Transposée d'une matrice. 333
2.11.12 Tenseurs batchs et tenseurs de hautes dimensions . . . 334
2.11.13 Calcul des valeurs propres d'une matrice hermitienne (dans un tenseur) 334
2.11.14 Extraction des termes diagonaux d'une matrice 335

3. Petit formulaire commenté 335
3.1 Définitions de différents types de matrices (et quelques usages) 335
3.2 Une formule intéressante concernant les déterminants 337
3.3 De la différentiation 338

4. À étudier plus avant 341
4.1 Méthodes d'algèbre linéaire et différentiation en algèbre linéaire 341
4.2 Algorithmes génétiques (ou GA) 341
4.3 Reinforcement Learning - RL. 343

5. Bibliographie du chapitre 345

Index 347

Chapitre 8
Simuler les attributs de l'intelligence

1. Identifier visuellement les objets

1.1 Le problème

Même de façon purement statique, qu'un artifact se déplace ou pas, l'identification des objets présents dans son contexte immédiat peut s'avérer très utile.

Comme pour un humain, cette identification peut se révéler incertaine ou imprécise. Après une phase de reconnaissance d'un objet, purement basée sur son aspect visuel et la détermination d'une probabilité associée, on a parfois besoin d'acquérir d'autres informations sur cet objet.

Au repérage global de la position de l'objet correspondant à une image captée dans une direction donnée, on voudra alors appliquer une deuxième passe de reconnaissance pour préciser divers attributs de l'objet qui pourraient par exemple être sa taille, sa position dans un espace 3D, la direction de son éventuel mouvement, sa texture…

On tentera de conforter la détermination des objets ou de leurs attributs en croisant diverses informations et en tenant compte de ce que l'on connaît du contexte de la prise de vue, de la direction de cette prise de vue et des évènements précédents.

Quand on dispose de plusieurs sources d'informations complémentaires, un système de reconnaissance visuelle par ailleurs peu précis peut suffire pour des applications robotiques opérationnelles.

Le travail en deux passes permet de converger vers une reconnaissance plausible. Il est par exemple peu probable de croiser un lion dans sa cuisine ! Ce qui permet d'éliciter plus rapidement l'option *chat* lors d'une reconnaissance visuelle qui donnerait une certaine probabilité à *lion* et une autre probabilité raisonnable à *chat*. Bien entendu, cette appréciation serait à reconsidérer si la direction de la prise de vue était dirigée vers la télévision du salon !

Que cela soit pour reconnaître visuellement des objets, ou pour d'autres prises de décision, il est souvent utile de faire appel à plusieurs technologies **en parallèle** ou les unes après les autres (**en série**).

Rappelons que l'on peut confier les décisions finales d'agrégation de l'opinion des différents algorithmes ou agents à des systèmes experts flous **FBRS** (*Fuzzy rules based System*) ou intégrer certains résultats via les **réseaux ANFIS** (*Adaptive network based fuzzy inference system*) que nous avons évoqués dans le premier chapitre.

1.2 Éléments de l'état de l'art

1.2.1 Les images labellisées

Pour pouvoir entraîner les modèles, divers contributeurs ont collectionné de grands ensembles d'images. À ces images ont été ajoutés des *labels*, c'est-à-dire la mention de leur contenu.

Parfois, plusieurs contenus sont labellisés sur une seule image et un opérateur a marqué l'endroit où se trouve chacun des objets identifiés. Ce marquage se fait classiquement via la mention d'une **box**, c'est-à-dire d'un rectangle qui entoure l'objet dans l'image et dont on récupère les coordonnées pendant l'entraînement.

On trouve également des images où le contour de l'objet a été déterminé de façon plus précise (comme avec un **lasso**), ce qui est beaucoup plus puissant, mais plus difficile à exploiter.

© Editions ENI - All rights reserved

Le problème de la détermination du contenu d'une image pour la faire labelliser (décrire) par un opérateur humain est en soi un problème plus compliqué qu'il n'y paraît, en effet la manière d'exprimer ce qu'elle voit est très variable d'une personne à l'autre. Pour décrire la présence d'un animal, on peut trouver des mentions aussi diverses que : chat, siamois…

Par ailleurs, on est fréquemment amené à déterminer les parties d'un objet (yeux, queue…) au sein d'un autre objet, ou des objets au sein d'une scène (chiens et chats jouant avec une balle), ou tout ensemble (la queue du chat de gauche) et même ajouter des attributs (un chat qui dort).

On comprend que deux façons de labelliser apparaissent : l'une se référant à une taxonomie bien identifiée (animal -> félin -> chat), l'autre en langage naturel ("un chat qui chasse en faisant semblant de dormir devant un trou de souris").

La première est la plus courante et nécessite peu de compétences en traitement du langage naturel. Elle permet de produire des bases de référence d'objet très fiables.

La deuxième permettra d'introduire une intelligence plus fine, contextualisée et le cas échéant émotionnelle dans l'apprentissage de la machine. Il existe d'ailleurs des datasets labellisant les sentiments évoqués dans des expressions humaines (joie, tristesse…).

Pour ce qui est des formes labellisées au lasso, la forme est parfois suggérée à l'opérateur par un traitement préliminaire sur l'image, qui identifie des contours ou des zones que l'on demandera à l'opérateur de labelliser.

Les packages comme **OpenCV**, que nous avons utilisé plus haut via l'API **Tensorlayers** au chapitre Améliorations du modèle, permettent en effet d'effectuer des traitements automatiques visant à extraire des zones significatives de l'image : la texture de la peau d'un chat est différente de la texture d'une voiture… donc apparaissent deux contours différents dans une image comportant un chat et une voiture que l'opérateur devra ensuite désigner comme tels.

Dans le même ordre d'idée, les zones colorées, de contrastes différents peuvent être proposées à des labellisations différentes.

Pour appréhender le cas des *box*, nous vous proposons d'étudier la façon dont a été construit l'*Open Images Dataset*, aujourd'hui dans sa version V5, au travers de l'article *The Open Images Dataset V4 : Unified image classification, object detection, and visual relationship detection at scale* (Kuznetsova et al. 2018).

La version actuelle étendue de ce dataset contient plus de **15 millions d'éléments tagués sur 600 catégories** et plus de 36 millions de labels au niveau image, étiqueter sur près de plus de 19 000 catégories. De plus, on y trouve plus de 390 000 annotations montrant des **relations entre labels de plus de 300 natures différentes**.

Voici quelques points de repère pour vous aider à naviguer dans ces données :

– Une catégorie possède un identifiant et un nom : /m/01nq_x , *Wild cat*
– Certaines catégories, pas toutes, peuvent déterminer une *box* comme :
  – /m/01yrx, *Cat*
  – /m/01lrl, *Carnivore*
  – /m/04rky, *Mammal*
– On trouve régulièrement la présence d'un héritage entre les catégories :

```
"LabelName": "/m/04rky",
"Subcategory": [
    {
        "LabelName": "/m/01lrl",
        "Subcategory": [

            {
                "LabelName": "/m/01yrx"
```

Il existe non seulement de nombreux datasets comportant des images pour nous aider, mais aussi de nombreux schémas de réseaux éprouvés.

© Editions ENI - All rights reserved

1.2.2 Les réseaux neuronaux phares de la reconnaissance d'objets dans des images

Les principaux objectifs assignés à ces réseaux sont :

– la détection d'objets (un ou n objets sont présents, le cas échéant au sein d'une box) ;
– la segmentation sémantique (chaque pixel est affecté à une forme qui à terme possédera un label) ;
– le sous-titrage d'images (exemple : "un chat siamois qui guette une souris blanche" : **captioning**).

Il est intéressant de **tester les différents modèles classiques** sur des cas précis, de préférence dans leur ordre d'apparition, afin de bien assimiler comment les différentes améliorations ont été acquises.

Dans le *fork* suivant https://github.com/laudehenri/ImageModels du *GitHub* de Dan Dixey, on trouve la définition rédigée en syntaxe Keras de certains modèles de référence ayant trait aux traitements d'images.

Il est aisé d'intégrer ces définitions dans votre code et d'expérimenter ces modèles dans vos contextes. Nous vous conseillons de les tester dans l'ordre de leur création :

1. AlexNet (2012, conçu pour 2 GPU) et son clone CaffeNet (une GPU) ;

2. ZFNet (2013) ;

3. VGG19 (2014, VGG16 possède souvent de meilleures performances) ;

4. GoogLeNet (aussi nommé Inception v1, 2014, qui utilise la convolution 1 X 1).

Par ailleurs, le code de ResNet (v1 en 2015 et versions suivantes) est parfaitement documenté dans le tutoriel de base de Keras (https://keras.io/examples/cifar10_resnet/). Il vous faudra le parcourir attentivement.

Une implémentation très instructive via Keras de DenseNet (Huang, Liu, and Weinberger 2016), où une couche reçoit les *feature maps* des couches précédentes, se trouve dans le *fork* suivant https://github.com/laudehenri/DenseNet des travaux de Somshubra Majumdar.

Enfin, le *fork* (https://github.com/laudehenri/keras-efficientnets) implémente la très intéressante idée de rationaliser l'évolution conjointe des dimensions d'un réseau quand on le met à l'échelle, c'est-à-dire le *scaling* tel qu'il est décrit dans un article très instructif (Tan and Le 2019).

Pour faire le point, une revue de ces réseaux, très didactique, est disponible ici : (Jing and Tian 2019).

1.2.3 Un exemple d'implémentation

Le package **imageai** permet d'accéder à divers modèles déjà entraînés et sauvés en *.h5*, y compris à vos propres modèles, cela dans de bonnes conditions de performance et avec une interface homogène entre les modèles.

C'est un atout important en termes de gain de temps de conception, mais aussi dans le cas où l'on veut facilement effectuer une identification s'appuyant sur plusieurs modèles à la fois (la technique consistant à compulser plusieurs prédictions dans un modèle qui en fera la synthèse est nommée **ensemble** dans le jargon des *data scientist*).

On peut accéder au package à cette adresse :
https://github.com/OlafenwaMoses/ImageAI/

ImageAI ne nécessite pas de nombreuses dépendances "exotiques" et fonctionne en utilisant TensorFlow comme *backend*. La ligne de commande pour l'installer est la suivante :

```
pip3 install https:// github.com/OlafenwaMoses/ImageAI/releases/download/2.0.3
/imageai-2.0.3-py3-none-any.whl
```

Dans le petit exemple suivant, **imageai** est utilisé avec un modèle très peu puissant, mais très compact et donc rapide.

```
from imageai.Prediction import ImagePrediction
import os
home_path = os.getcwd()


p = ImagePrediction()

# utilisation de Squeezenet, le setModel permet de définit le type de modèle
p.setModelTypeAsSqueezeNet()
p.setModelPath(os.path.join(home_path, "squeezenet_weights_tf_dim_ordering_tf
_kernels.h5"))
```

© Editions ENI - All rights reserved