intelligence artificielle pour débutants (3)

Créez votre propre réseau de neurones

Walter Trojan (Allemagne)

Dans les deux articles précédents de cette série, j'ai présenté la carte Maixduino et montré comment avec MicroPython mettre en œuvre un réseau de neurones prêt à l'emploi pour reconnaître les visages dans les images. Ce troisième et dernier épisode vous propose de créer votre propre réseau neuronal. Il traite également de l'ESP32 qui, en tant que coprocesseur sur la carte, prend en charge les tâches de communication.

J'ai fait jusqu'ici sur le Maixduino des démonstrations d'intelligence artificielle (IA) déjà formée. Si vous souhaitez créer vos propres réseaux neuronaux (que, pour faire court, nous appellerons NN pour *neuronal network*), vous devrez passer par le stade du développement et de la *formation* de votre intelligence artificielle sur un PC ou un service dématérialisé, puis transférer vers Maixduino votre NN une fois formé. Qu'il faille passer par cette procédure n'est pas un inconvénient, car la formation de votre NN nécessite de toute manière à la fois de grandes quantités de données et une grande puissance de traitement. Une fois votre NN formé, il sera utilisable sur une plateforme mobile, avec une faible consommation d'énergie, p. ex. une commande de robot.
Je décrirai donc le chemin qui mène de la conception d'un NN à son utilisation sur le Maixduino. Tout d'abord, cela nécessite sur le PC un environnement de développement typique sous Linux (**fig. 1**).

Kit d'IA

Comme interface avec le développeur, nous aurons l'infrastructure d'intelligence artificielle Keras, le Lego de l'intelligence artificielle. Elle s'appuie sur l'infrastructure bien éprouvée *TensorFlow*, qui fait tout le travail. Il existe aussi un système alternatif comme Theano. Si le PC est équipé d'une carte graphique Nvidia, vous pouvez utiliser les nombreux cœurs GPU via la bibliothèque CUDA, sinon vous mettez les CPU du PC en action avec BLAS.

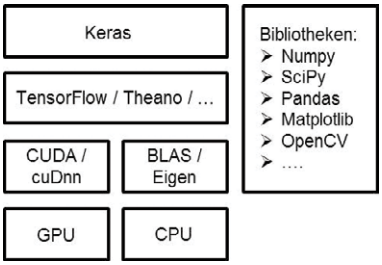


Figure 1 : Structure d'un environnement populaire de développement de l'IA.

Voici un récapitulatif des principaux éléments constitutifs :

- *Keras* : infrastructure d'IA, basée sur *TensorFlow* ou autre infrastructure
- *TensorFlow* : infrastructure d'IA puissante et répandue, conçue et fournie par Google
- *CUDA / cuDnn* : bibliothèque de support des GPU Nvidia
- *BLAS / Own* : bibliothèque de programmes pour des opérations élémentaires d'algèbre linéaire comme la multiplication de vecteurs et de matrices.

Les bibliothèques suivantes sont intégrées selon les besoins :

- *Numpy* : fournit des fonctions très efficaces, en particulier pour les opérations matricielles
- *SciPy* : bibliothèque basée sur *Numpy* avec des fonctions supplémentaires, p. ex. des équations différentielles
- *Pandas* : tableur utilisable dans le programme
- *Matplotlib* : dessine des diagrammes et des illustrations
- *OpenCV* : traitement d'images puissant
- *HDF5* : bibliothèque pour le traitement et le stockage d'ensembles de données hétérogènes
- *Graphviz* : visualisation d'informations structurées
- *pydot-ng* : utilitaire pour Graphviz
- *Jupyter* : environnement de développement pour Python, les sections de programme sont structurées en cellules exécutables individuellement, comme le préconisent les professionnels

D'autres infrastructures d'IA sont disponibles, p. ex. Theano, Torch, Caffe, Pytorch, Yolo avec bien sûr d'autres bibliothèques. Cela témoigne de la richesse du monde Linux/Python et justifie des recherches sur Google. L'environnement (fig. 1) peut être configuré sur un PC sous Linux à l'aide des commandes de terminal suivantes :

Mise à jour :
```
sudo apt-get update
sudo apt-get upgrade
```

Python et pip :
```
sudo apt-get install python3-pip python3-dev
pip install --upgrade pip
```

OpenBlas :
```
sudo apt-get install build-essential cmake git unzip
pkg-config libopenblas-dev liblapack-dev
```

SciPy, Numpy :
```
sudo apt-get install python-numpy python-scipy python-yaml
```

matplotlib :
```
sudo pip3 install matplotlib
```

HDF5 :
```
sudo apt-get install libhdf5-serial-dev python-h5py
```

Graphviz, pydot-ng :
```
sudo apt-get install graphviz
sudo pip3 install pydot-ng
```

OpenCV :
```
sudo apt-get install python3-opencv
# vers. 3, la procédure de la v.4 est complexe
```

Tensorflow 2.0 :
```
pip3 install tensorflow==2.0.0b1
```

Keras :
```
sudo pip3 install keras
```

Pandas :
```
sudo apt-get install python3-pandas
```

Thonny :
```
sudo apt-get install thonny
```

Jupyter :
```
pip3 install jupyter
```

Appel :
`jupyter notebook` (commande du terminal)

Nous avons là nos éléments de base. S'il faut des composants supplémentaires, spécifiques au projet, ils seront installés avec `pip` ou `apt-get`. L'éditeur Python choisi, *Thonny*, se contente de fonctions minimales mais suffisantes. Les pros en utilisent d'autres comme Jupyter, qui permet de structurer les programmes en cellules exécutables individuellement. Tout est prêt pour passer au développement d'un NN à partir de zéro puis pour le transférer sur Maixduino. Je propose comme exemple la reconnaissance de chiffres manuscrits. La base de données MNIST (*Modified National Institute of Standards and Technology database*) en est la base, avec 60.000 échantillons d'images disponibles pour la formation et 10.000 pour les tests. Toutes les images au format 28 x 28 pixels ont une étiquette avec la valeur numérique correcte (**fig. 2**).

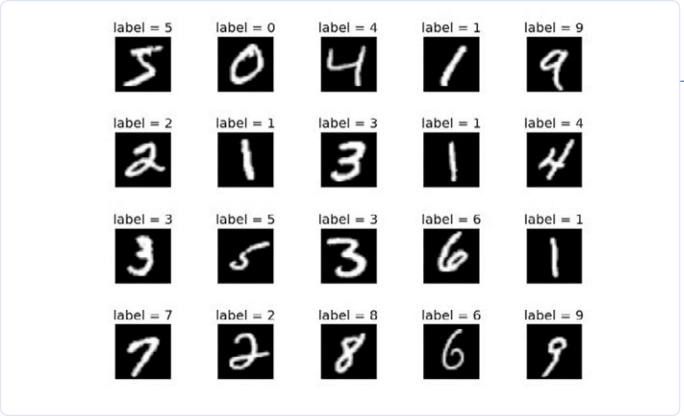


Figure 2. Extrait de la base de données du MNIST.

À petits pas vers le NN

Le NN que nous obtiendrons avec Keras/Tensorflow est destiné à classer correctement des chiffres manuscrits encore inconnus de lui. En principe, le développement d'un NN se déroule comme suit :

- **préparation des données** : convertir les données de formation et de test en un format approprié
- **définition** : définition de la structure NN avec le type et le nombre de couches
- **compilation** : traduction des instructions Keras en instructions Tensorflow
- **entraînement** : reprise répétée des données d'entraînement pour ajuster la pondération NN
- **test** : vérifiez l'exactitude de la reconnaissance à l'aide des données du test.
- **reconnaissance** : présentation de nouveaux chiffres et classification

Préparation des données

Les données du MNIST disponibles dans la bibliothèque Keras sont importées dans le format approprié.

Définition

Le modèle suivant, mis au point par des professionnels, présente un excellent taux de détection de plus de 99 %.

```
model = Sequential()
     # modèle séquentiel, il en existe de
     # fonctionnels
model.add(Conv2D(32, kernel_size=(3, 3),
     # première couche de convolution
     # bidimensionnelle
       avec 32 sorties
                activation='relu',
                   # fonction d'activation
                input_shape=input_shape))
                   # format d'entrée 28*28
model.add(Conv2D(64, (3, 3), activation='relu'))
     # deuxième couche de convolution avec 64 sorties
       fonction d'activation relu
model.add(MaxPooling2D(pool_size=(2, 2))
     # densification
model.add(Dropout(0.25))
     # renforcement de la robustesse
model.add(Flatten())
     # convertir de deux à une dimension
model.add(Dense(128, activation='relu'))
     # troisième couche NN avec 128 sorties et
```

ESP32 - AIDE POUR LES TÂCHES SPÉCIALES

La puce d'intelligence artificielle K210, par ailleurs bien dotée, manque d'entrées analogiques et ne communique ni par WiFi ni par Bluetooth. Une brèche que comble l'ESP32 qui, outre ces fonctions, dispose d'un double processeur avec horloge de 240 MHz. Le K210 et l'ESP32 sont couplés par une connexion SPI rapide◦ ; six entrées analogiques sont acheminées vers le connecteur femelle Arduino. Une connexion série à ESP32 est possible via le port *ttyUSB1*, et l'ESP32 est programmé via le K210 en utilisant l'EDI MaixPy.

Voici un premier exemple pour l'acquisition de signaux analogiques :

```
import network
   # importation des modules requis
import utime
from Maix import GPIO
from fpioa_manager import *

#iomap at MaixDuino
   # enregistrement des GPIO pour l'interface ESP32
fm.register(25,fm.fpioa.GPIOHS10)    # cs
fm.register(8,fm.fpioa.GPIOHS11)     # rst
fm.register(9,fm.fpioa.GPIOHS12)     # rdy
fm.register(28,fm.fpioa.GPIOHS13)    # mosi
fm.register(26,fm.fpioa.GPIOHS14)    # miso
fm.register(27,fm.fpioa.GPIOHS15)    # sclk

# definition of network interface
nic = network.ESP32_SPI(cs=fm.fpioa.GPIOHS10,rst=fm.fpioa.GPIOHS11,rdy=fm.fpioa.GPIOHS12,
mosi=fm.fpioa.GPIOHS13,miso=fm.fpioa.GPIOHS14,sclk=fm.fpioa.GPIOHS15)

adc = nic.adc( (0,1,2) )
    # résultats de ADC0 ADC1 ADC2
print('ADC 0,1,2')
    # impression des résultats de ADC 0-2
print(adc)
print()
print('ADC 0,1,2,3,4,5')

while True:
    try:
        adc = nic.adc()
    # résultat ADC0-5
    except Exception as e:
        print(e)
           # en cas d'erreur imprime la cause
        continue
    for v in adc:
        print("%04d" %(v), end=" ")
           # impression des résultats formatés de ADC 0-5
    print()
    utime.sleep_ms(1000)
           # pause 1000 ms
```

Au début du programme, les modules Python requis sont importés, puis les GPIO pour la communication avec ESP32 sont enregistrés et affectés à l'objet `nic.` Pour enregistrer les entrées analogiques, la fonction `nic.adc` reçoit un tuple avec les canaux

```
    # activation relu
model.add(Dropout(0.5))
    # renforcement de la robustesse
model.add(Dense(num_classes, activation='softmax'))
    # quatrième couche NN avec 10 sorties et
    # activation softmax
```

Le NN dispose de quatre couches, deux convolutives et deux classiques (*Dense*) avec un nombre de nœuds (32/64/128/10) qui reste intelligible.

L'activation de la dernière couche avec `softmax` limite la somme des valeurs initiales à la probabilité 1. Dans les deux premières couches, les détails tels que les lignes, les arcs et les points sont filtrés dans des champs de 3 x 3 pixels chacun. Entre les couches sont insérés divers filtres :

- `MaxPooling` compresse le résultat d'une couche et ne transfère que les valeurs moyennes d'un champ (ici 2 x 2). `Dropout` néglige

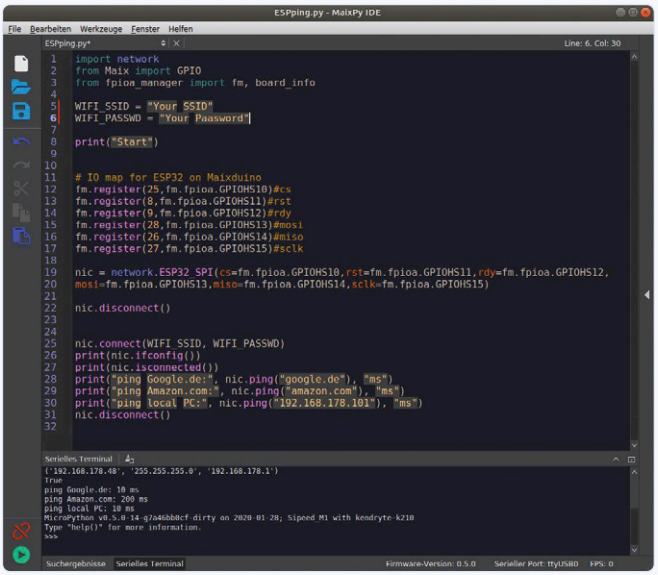
souhaités, ici ADC0 à ADC2. Après l'affichage de ces résultats, les six entrées disponibles sont même interrogées dans la boucle `while` sans nommer les numéros de canaux. Si des erreurs se produisent lors de l'acquisition, le code d'erreur est donné par la fonction d'exception.

Pour vérifier le fonctionnement de l'acquisition analogique, j'ai alimenté un niveau d'entrée variable au canal ADC0 via un potentiomètre. Veuillez noter que l'entrée analogique native ne couvre que la plage de 0 à 1,0 V. Les valeurs affichées par le terminal confirment le bon fonctionnement :

```
ADC 0,1,2
(0, 1786, 73)

ADC 0,1,2,3,4,5
0000 1469 0082 0521 0120 0001
    # potentiomètre à 0 V
0000 1813 0160 0606 0291 0000
2622 2135 0210 0630 0323 0000
    # potentiomètre à mi-course
4095 2421 0259 0575 0261 0000
4095 2472 0274 0615 0315 0000
    # potentiomètre à 1,0 V
```

Les autres canaux n'étaient pas connectés et affichent des valeurs aléatoires dans les colonnes 1 à 5.



Un deuxième exemple illustre la communication sur l'nternet◦ : deux portails bien connus, Google et Amazon, ainsi qu'un ordinateur local sont interrogés (voir la capture d'écran).

L'importation des modules ainsi que l'enregistrement des ports GPIO et la définition de l'objet `nic` se font comme dans le premier exemple. Pour utiliser l'internet, les données d'accès au routeur WiFi sont bien sûr nécessaires. Après la connexion au réseau, l'adresse IP de l'ESP32 peut être demandée avec `nic.ifconfig()` et l'état de la connexion est affiché avec `nic.isconnected()`. Avec la commande `nic.ping()`, seule l'adresse de destination doit être spécifiée. Outre les adresses internet, les ordinateurs locaux peuvent également être *pingés* à leur adresse IP.

Il existe de nombreux autres exemples d'application, par exemple un serveur web pour la transmission de l'analyse IA, une collection d'objets provenant d'Internet, et bien d'autres encore. Laissez-vous inspirer par les exemples présentés sous le lien [5] et n'hésitez pas à expérimenter. Veuillez noter que Maixduino est un encore jeune et dans une phase de développement assez mouvementée, c'est-à-dire qu'en plus du micrologiciel du K210, une mise à jour est également nécessaire de temps en temps pour la connexion de l'ESP32. Cela peut être fait avec la séquence de commandes suivante après avoir téléchargé le fichier binaire mentionné ci-dessous [4] :

```
pip install esptool
esptool.py --chip esp32 --port /dev/ttyUSB1 erase_flash
esptool.py --chip esp32 --port /dev/ttyUSB1 --baud 1500000 write_flash -z 0x0000
   maixduino_esp32_firmware_v1.4.0.bin
```

des nœuds choisis au hasard pendant la formation et attribue ainsi plus d'importance aux nœuds voisins, ce qui augmente la robustesse du NN. `Flatten` transforme une sortie multidimensionnelle de CNN en sortie unidimensionnelle.

Compilation

Dans cette phase a lieu la conversion des instructions Keras en instructions Tensorflow.

```
model.compile(loss=keras.losses.categorical_
   crossentropy,   # attribution de la fonction de
   perte
       optimizer=keras.optimizers.Adadelta(),
           # sélection de l'optimiseur
              metrics=["accuracy"])
                   # définition de la précision
```

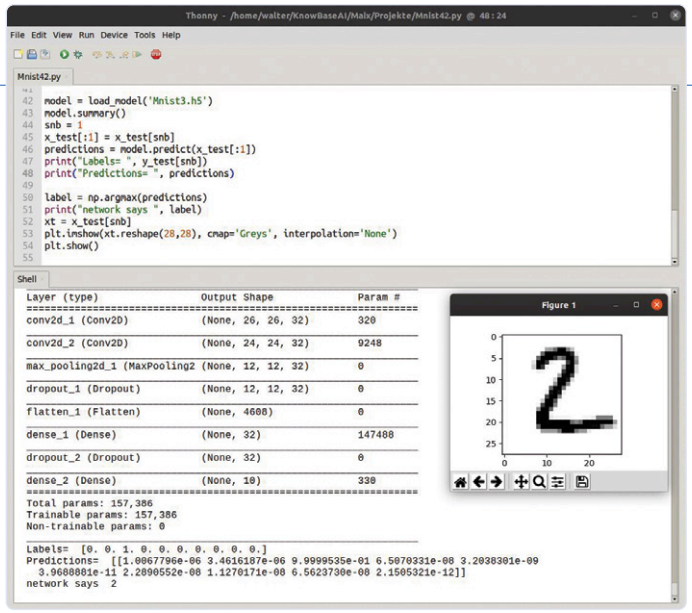


Figure 3. Reconnaissance du chiffre 2.

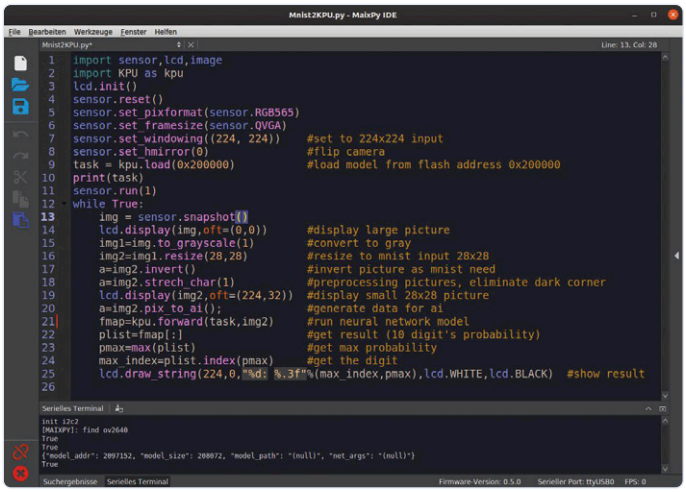


Figure 4. Reconnaissance de chiffres avec MicroPython.

Formation

Ici l'entraînement du NN avec 60.000 symboles numériques se joue en 12 manches (époques).

```
model.fit(x_train, y_train,
      # x_train contient des images, y_train contient
      # des étiquettes
            batch_size=batch_size,
               # la formation se fait par lots,
               # p. ex. par paquets de 16
            epochs=epochs,
              # nombre d'epochs, manches avec toutes
              # les données
            verbose=1,
              # résultat de la formation
            validation_data=(x_test, y_test))
              # validation du modèle avec les données
              # de test
```

Test

Enfin, le taux de détection est déterminé à l'aide des 10.000 données du test.

```
score = model.evaluate(x_test, y_test, verbose=0)
model.save("Mnist3.h5")
```

Le modèle peut maintenant être sauvegardé avec la structure et les poids formés au *format h5* et utilisé plus tard – également sur une autre plateforme – pour la reconnaissance.

Détection

La détection se fait localement, sur Maixduino après le transfert, mais vous pouvez déjà la faire sur le PC sous Linux avec l'éditeur Thonny (**fig. 3**).
On charge le NN formé pour qu'il reconnaisse de nouveaux chiffres manuscrits ; ensuite sera produite la structure du modèle. Le lecteur attentif aura remarqué que dans certaines couches, le nombre de nœuds a été réduit pour que le NN tienne dans la mémoire plus exiguë (max. 5,9 Mo ici). Ensuite, le deuxième élément des données d'essai est reconnu, par hasard un «2», comme le confirme l'étiquette de cet ensemble de données. Le NN voit les choses de la même manière, car sa reconnaissance donne une probabilité de 0,9999 pour cet indice, alors que tous les autres chiffres obtiennent des valeurs inférieures et négligeables. En ne retenant que la valeur la plus grande du tableau, *numpy,* avec la fonction `argmax` contribue à produire une évaluation compréhensible. Par ailleurs, l'image à reconnaître est dessinée grâce à la bibliothèque *matplotlib*. La formation et le test sur le PC sous Linux sont achevés. Il reste le transfert vers le Maixduino, avec les étapes suivantes :

- créer le modèle avec formation et test à l'aide de Keras, le résultat est un modèle d'IA au format *h5* (comme indiqué ci-dessus)
- convertir le *modèle h5* en TensorflowLite au format *tflite*
- compiler le modèle *tflit* avec le compilateur *nncase* en un format pour la KPU, le *format kmodel*
- flasher ce modèle dans Maixduino en utilisant Kflash à l'adresse 0x200000
- créer un script Python en utilisant l'IDE MaixPy et le télécharger sur Maixduino pour l'exécuter et le tester

Vous trouverez une documentation supplémentaire à ce sujet dans le dossier du projet [3], où tous les programmes et fichiers utilisés dans l'article sont annotés et commentés.
La figure 4 montre l'application sur le Maixduino.

Après l'importation des modules requis, l'appareil photo et l'écran sont initialisés et ajustés. Ensuite, le NN de l'adresse 0x200000 est chargé et est disponible avec l'objet `task`. Dans la boucle `while` une image est d'abord capturée puis affichée sur l'écran. Ensuite, l'image est préparée pour le NN : Conversion en gris, conversion en 28 x 28 pixels et inversion pour le MNIST. Après avoir ajusté les valeurs des pixels à un format utilisable par l'IA (généralement division par 255), l'image est transférée au NN pour reconnaissance. Celui-ci renvoie son résultat dans la variable `fmap`. Ensuite apparaissent sur l'écran le chiffre reconnu et la probabilité de succès de cette reconnaissance. Vite faite bien faite, une reconnaissance de chiffres en 25 commandes !
Le test pratique (**fig. 5**) confirme le bon fonctionnement de notre NN et, avec la reconnaissance de quelques images par seconde, la performance de la KPU.

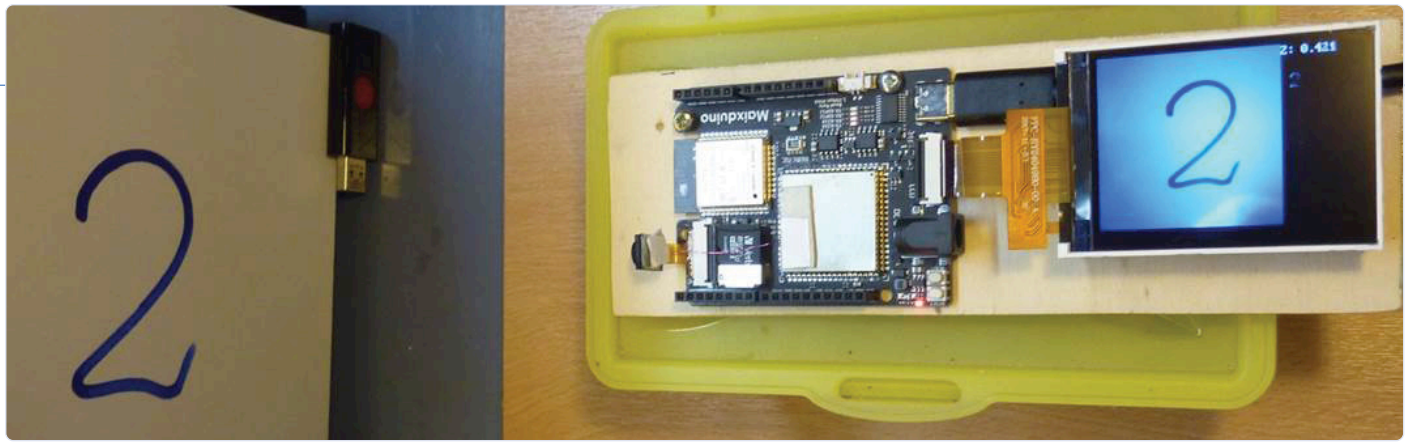


Figure 5. Configuration de test de la reconnaissance de chiffres avec Maixduino.

Ce n'est que le début !

Le matériel puissant et l'environnement logiciel aujourd'hui disponible montrent à quel point Maixduino est commode pour se frotter à l'intelligence artificielle. Grâce à sa faible consommation, il peut être utilisé dans des appareils mobiles avec des réseaux neuronaux déjà formés. Pour faire du développement, de la formation et de l'exécution sur une même plateforme, on préférera des systèmes plus trapus, avec Linux installé, p. ex. le Nvidia Jetson Nano avec sa flopée de logiciels et un support solide. Consultez éventuellement les nouvelles versions comme Rock Pi N10 et d'autres, mais avant d'acheter assurez-vous de la disponibilité du logiciel et d'une documentation de qualité. Et n'oubliez pas que pour commencer un PC sous Linux, avec ou sans carte graphique GTX, fait tout aussi bien.

Ça bouge dans l'IA, et ce n'est que le début. Cette petite série n'avait pour ambition que de présenter une partie de la partie visible de l'iceberg. Car outre la méthode d'*apprentissage supervisé* présentée ici, dans laquelle un réseau neuronal est formé avec des données connues dites *étiquetées*, il existe bien d'autres solutions d'IA.

Dans l'*apprentissage non supervisé,* des données inconnues sont introduites dans le réseau neuronal et celles-ci sont divisées en grappes, c'est-à-dire en groupes similaires les uns aux autres, en fonction de leurs caractéristiques. Cette méthode est utilisée, par exemple, pour l'analyse de grands ensembles de données afin d'identifier certains points communs. Ou, dans le cas des *autocodeurs,* pour la suppression du bruit dans la reconnaissance des caractères.

Vous êtes-vous déjà demandé comment il est possible que les NN se révèlent meilleurs que les humains dans certains domaines et puissent même les battre au jeu de Go alors que les données d'apprentissage leur ont été fournies par des humains ? Il existe pour cela un autre type d'IA, l'*apprentissage renforcé*, dans lequel le NN est motivé par un système de récompense dans lequel les résultats sont améliorés en combinant plusieurs étapes d'analyse. Cette méthode est utilisée dans le secteur financier et pour les jeux informatiques, entre autres. Un des points faibles de l'IA est à l'heure actuelle l'architecture de la «boîte noire» : après la saisie des données, l'IA effectue une classification sans la justifier. On ne sait toujours pas, par exemple, pourquoi l'IA suggère à un cabinet de recrutement tel candidat plutôt que tel autre. On a ainsi constaté que tel système préférait les hommes, simplement parce dans les données de formation de l'IA les hommes étaient plus nombreux que les femmes. Ces faibles sont l'objet de recherches approfondies pour tenter de rendre *explicable* l'*IA :* bientôt les NN justifieront leurs décisions qui deviendront ainsi (plus) compréhensibles pour les utilisateurs.

Vous aussi pouvez participer à ce vaste mouvement, en vous aidant des vidéos, des tutoriels et de la documentation technique disponibles. Vous pourrez bientôt ajouter une bonne part d'intelligence à vos propres projets. Un dernier conseil, car ça ne manquera pas de vous paraître complexe et même difficile : n'abandonnez jamais !

200023-C-02

LIENS

[1] **intelligence artificielle pour débutants (1), Elektor mai/juin 2020** : www.elektormagazine.fr/200023-01
[2] **intelligence artificielle pour débutants (2), Elektor juillet-août 2020** : www.elektormagazine.fr/200023-B-02
[3] **intelligence artificielle pour débutants (3) - cet article** : www.elektormagazine.fr/200023-C-02
[4] **progiciel ESP32** : https://github.com/sipeed/Maixduino_esp32_fimware/releases/tag/v1.4.0
[5] **programmes de démonstration ESP** : https://github.com/sipeed/MaixPy_scripts/tree/master/network