Walid GHETTAS EISE-4

Compte-rendu TP2: Framework Keras

Keras est une API de haut niveau permettant de créer des modèles d'apprentissage en profondeur. Au cours de ce TP, nous allons apprendre à utiliser cet outil afin de réaliser des modèles de régression, de classification et de convolution.

Résistance du béton

A partir de données concernant la résistance à la compression de différents échantillons de béton en fonction des volumes des différents matériaux utilisés pour leur fabrication, nous mettons en place un modèle de régression.

Nous téléchargeons et lisons avec la bibliothèque pandas à l'aide des commandes indiquées sur le sujet.

Après avoir vérifié que nous avons bien 1030.9 points de données contenu pour le jeu de donnée avec la fonction shape, nous vérifions l'ensemble de données pour toutes les valeurs manquantes avec les fonctions suivantes avant de scinder les données en prédicteurs et cibles (predictors and target).
Ici, nous avons séparé les données en 2 parties : une pour nos données de prédictions et une pour nos données de cibles sur la résistance.

```
   Cement  Blast Furnace Slag  Fly Ash  ...  Coarse Aggregate  Fine Aggregate  Age
0   540.0                0.0      0.0  ...            1040.0           676.0   28
1   540.0                0.0      0.0  ...            1055.0           676.0   28
2   332.5              142.5      0.0  ...             932.0           594.0  270
3   332.5              142.5      0.0  ...             932.0           594.0  365
4   198.6              132.4      0.0  ...             978.4           825.5  360

[5 rows x 8 columns]
0    79.99
1    61.89
2    40.27
3    41.05
4    44.30
Name: Strength, dtype: float64
```

(Résultat sur le contrôle contrôle de validité des prédicteurs et des données cibles)

Nous faisons un rapide contrôle de validité des prédicteurs et des données cibles, puis nous normalisons les données en soustrayant la moyenne et en divisant par l'écart type. Nous aurons ainsi nos variables à la fois centrées et réduites, le but étant de minimiser le taux d'erreur en sortie.

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
0	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	0.862735	-1.217079	-0.279597
1	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	1.055651	-1.217079	-0.279597
2	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	3.551340
3	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	5.055221
4	-0.790075	0.678079	-0.846733	0.488555	-1.038638	0.070492	0.647569	4.976069

(Résultats normalisés)

Nous décidons d'importer Keras et définissons une fonction qui définit notre modèle de régression afin que nous puissions l'appeler facilement pour créer notre modèle. Le modèle de ce réseau de neurone contient 1 couche d'entrée avec 7 entrées (n_cols input), 2 couches cachées de 30 neurones et une valeur de sortie qui correspond à la prédiction sur la résistance du béton.

```
def regression_model(): #define regression model
    model = Sequential() # create model
    model.add(Dense(30, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(30, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error') # compile model
    return model
```

Après la création de notre modèle, nous entraînons notre modèle à l'aide des commande indiqué dans le sujet.

```
model = regression_model()
model.fit(predictors_norm, target, validation_split=0.3, epochs=100, verbose=2)
```

Predictors_norm sont les données de prédictions normalisés.
Target sont les résultats attendus.
Validation_split indique avec une valeur de 0,3 que 70 % des données sont utilisées pour l'apprentissage et 30 % des données pour les tests.
Epochs est le paramètre qui définit le nombre de fois que l'algorithme d'apprentissage fonctionnera dans l'ensemble du jeu de données d'apprentissage. Une époque signifie que chaque échantillon de l'ensemble de données d'apprentissage a eu l'occasion de mettre à jour les paramètres du modèle interne.
Verbose permet d'afficher plus ou moins d'informations pendant la phase d'apprentissage. Concernant les paramètres de sortie affichés pendant l'apprentissage, le loss est l'erreur entre les valeurs prédites par le réseau et la valeur réelle sur le jeu de donnée d'entraînement, tandis que val_loss est l'erreur entre les valeurs prédites et la valeur réelle du jeu de données de test.

A la suite de différents test, nous remarquons que plus on a de neurones, plus les valeurs réelle sont celles prédites (loss diminue). Ce qui est curieux c'est qu'à l'inverse, le val_loss qui n'avait pas tellement évolué jusque là connait cette fois-ci un changement non négligeable.

Pic d'ozone

Nous allons nous intéresser cette fois-ci à un autre tout en applicant le même cheminement de travail que pour la résistance du béton. Il s'agit d'étudier des données météorologiques afin d'établir un modèle de régression permettant de prédire les pics d'ozone.
Nous commençons par remplacer les données textuelles par un code numérique :

- Pour la pluie : Pluie = 1 et Sec = 2
- Pour le vent : Est= 1, Sud = 2, Ouest =3 et Nord= 4

Désormais la colonne maxO3 est cible. Nous réalisons maintenant un modèle de régression avec 1 couche avec 13 entrées avec 13 entrées, 2 couches cachées de 300 neurones et une valeur de sortie qui correspond à la prédiction sur le pic d'ozone.

Résultat obtenu:

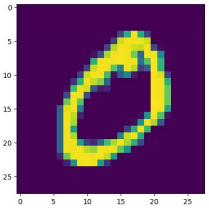
Epoch 1/100
- 0s - loss: 9521.0198 - val_loss: 7230.6359

Epoch 100/100
- 0s - loss: 2260.0318 - val_loss: 2628.7411

Les valeurs loss et val_loss sont très élevées, mais en rajoutant une couche de neurones la valeur de loss est diminue mais par opposition au val_loss qui va avoir tendance a augmenté.

Jeu de données d'images, MINST

Ici, nous essayons de découvrir le chiffres exactes écrits à la main. Ainsi, comment précédemment nous commençons par charger les données.
Avec X, nos entrées et Y, notre cible.



(1ère image de l'ensemble d'apprentissage en utilisant la fonction imshow() de Matplotlib)

Comme dit dans le sujet, avec les réseaux de neurones conventionnels, nous ne pouvons pas alimenter l'image en entrée telle quelle. Nous allons donc aplatir nos images et les transformer en vecteurs unidirectionnels, elles vont passer d'une résolution de 28 x 28 à 1 x 784. Les valeurs des pixels pouvant aller de 0 à 255, nous normalisons les vecteurs entre 0 et 1.

```
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255
```

Enfin, avant de commencer à construire notre modèle, on rappelle que pour la classification, nous devons diviser notre variable cible en catégories. Nous utilisons la fonction to_categorical() du paquet Keras Utilities.

```
# one hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

Définissons maintenant une fonction qui définit notre modèle de classification afin que nous puissions l'appeler facilement pour créer notre modèle. Le modèle de ce réseau de neurone contient 1 couche d'entrée du nombre de pixels par images (num_pixels input), 1 couche cachée de 100 neurones et une couche de sortie avec les 10 classes (num_classes) obtenu grâce à la fonction d'activation softmax.

```
# define classification model
def classification_model():
 # create model
 model = Sequential()
 model.add(Dense(num_pixels, activation='relu', input_shape=(num_pixels,)))
 model.add(Dense(100, activation='relu'))
 model.add(Dense(num_classes, activation='softmax'))
 # compile model
 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
 return model

model = classification_model()
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, verbose=2)
scores = model.evaluate(X_test, y_test, verbose=0)

print('Accuracy: {}% \n Error: {}'.format(scores[1], 1 - scores[1]))

model.save('classification_model.h5')
```

Nous créons alors le modèle avant de l'entraîner, de l'évaluer et et afficher les taux de précision et d'erreur du modèle.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 24s - loss: 0.1852 - accuracy: 0.9438 - val_loss: 0.1084 - val_accuracy: 0.9655
Epoch 2/10
 - 26s - loss: 0.0769 - accuracy: 0.9762 - val_loss: 0.0817 - val_accuracy: 0.9743
Epoch 3/10
 - 26s - loss: 0.0539 - accuracy: 0.9827 - val_loss: 0.0921 - val_accuracy: 0.9742
Epoch 4/10
 - 25s - loss: 0.0409 - accuracy: 0.9866 - val_loss: 0.0619 - val_accuracy: 0.9821
Epoch 5/10
 - 26s - loss: 0.0302 - accuracy: 0.9905 - val_loss: 0.0772 - val_accuracy: 0.9798
Epoch 6/10
 - 26s - loss: 0.0258 - accuracy: 0.9918 - val_loss: 0.0709 - val_accuracy: 0.9814
Epoch 7/10
 - 26s - loss: 0.0228 - accuracy: 0.9926 - val_loss: 0.0893 - val_accuracy: 0.9771
Epoch 8/10
 - 25s - loss: 0.0199 - accuracy: 0.9933 - val_loss: 0.0883 - val_accuracy: 0.9803
Epoch 9/10
 - 25s - loss: 0.0179 - accuracy: 0.9944 - val_loss: 0.0890 - val_accuracy: 0.9796
Epoch 10/10
 - 26s - loss: 0.0154 - accuracy: 0.9952 - val_loss: 0.0929 - val_accuracy: 0.9819
Accuracy: 0.9818999767303467%
 Error: 0.01810002326965332
```

Entre 5 et 10 minutes ont été nécessaires pour recueillir ces résultats. L'entraînement de notre modèle à chaque utilisation est impossible. Ainsi, on utilise la fonction save() pour enregistrer le modèle après un apprentissage. et load_model() lorsque l'on est prêt à utiliser le modèle à nouveau.

Jeu de données FASHION_MNIST

Réalisons un modèle de classification sur le jeu de données FASHION-MNIST. Les images à déterminer avec précision sont des images d'habits. images de vêtements. Nous suivons le même processus que dans précédemment.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 21s - loss: 0.4717 - accuracy: 0.8295 - val_loss: 0.4065 - val_accuracy: 0.8544
Epoch 2/10
 - 21s - loss: 0.3588 - accuracy: 0.8691 - val_loss: 0.4067 - val_accuracy: 0.8494
Epoch 3/10
 - 21s - loss: 0.3219 - accuracy: 0.8822 - val_loss: 0.3415 - val_accuracy: 0.8790
Epoch 4/10
 - 20s - loss: 0.2987 - accuracy: 0.8894 - val_loss: 0.3545 - val_accuracy: 0.8717
Epoch 5/10
 - 20s - loss: 0.2795 - accuracy: 0.8960 - val_loss: 0.3323 - val_accuracy: 0.8810
Epoch 6/10
 - 20s - loss: 0.2634 - accuracy: 0.9005 - val_loss: 0.3331 - val_accuracy: 0.8837
Epoch 7/10
 - 20s - loss: 0.2506 - accuracy: 0.9063 - val_loss: 0.3330 - val_accuracy: 0.8847
Epoch 8/10
 - 21s - loss: 0.2394 - accuracy: 0.9091 - val_loss: 0.3307 - val_accuracy: 0.8841
Epoch 9/10
 - 19s - loss: 0.2283 - accuracy: 0.9120 - val_loss: 0.3448 - val_accuracy: 0.8819
Epoch 10/10
 - 20s - loss: 0.2203 - accuracy: 0.9161 - val_loss: 0.3310 - val_accuracy: 0.8878
Accuracy: 0.8877999782562256%
 Error: 0.11220002174377441
```

Par comparaison avec MNIST, une erreur plus grande de 0,1 et la précision a diminué légèrement.

Réseau de neurones convolutionnel avec Keras (1 couche caché)

Apprenons maintenant à utiliser la bibliothèque keras pour construire des réseaux de neurones convolutionnels. Nous allons également utiliser le populaire jeu de données MNIST et comparer nos résultats à ceux d'un réseau de neurones conventionnel. Nous conservons les différentes étapes (chargement des données, on aplatit les images en vecteurs unidimensionnel, normalisation des vecteurs, on catégorise nos données cibles...). Nous redéfinissons une fonction qui crée notre modèle. Nous remarquons que notre modèle contient une couche de convolution avec 16 neurones et d'une couche MaxPooling2D qui joue un rôle aplatissement des données. Enfin, on appelle la fonction pour créer le modèle, puis on le forme et on l'évalue de la même manière que précédemment.

```
Epoch 1/10
 - 13s - loss: 0.2808 - accuracy: 0.9223 - val_loss: 0.0892 - val_accuracy: 0.9721
Epoch 2/10
 - 11s - loss: 0.0799 - accuracy: 0.9764 - val_loss: 0.0648 - val_accuracy: 0.9796
Epoch 3/10
 - 11s - loss: 0.0550 - accuracy: 0.9841 - val_loss: 0.0610 - val_accuracy: 0.9812
Epoch 4/10
 - 11s - loss: 0.0422 - accuracy: 0.9877 - val_loss: 0.0426 - val_accuracy: 0.9862
Epoch 5/10
 - 11s - loss: 0.0352 - accuracy: 0.9893 - val_loss: 0.0379 - val_accuracy: 0.9871
Epoch 6/10
 - 11s - loss: 0.0286 - accuracy: 0.9914 - val_loss: 0.0393 - val_accuracy: 0.9853
Epoch 7/10
 - 11s - loss: 0.0238 - accuracy: 0.9926 - val_loss: 0.0379 - val_accuracy: 0.9871
Epoch 8/10
 - 14s - loss: 0.0199 - accuracy: 0.9938 - val_loss: 0.0359 - val_accuracy: 0.9877
Epoch 9/10
 - 13s - loss: 0.0169 - accuracy: 0.9948 - val_loss: 0.0358 - val_accuracy: 0.9879
Epoch 10/10
 - 12s - loss: 0.0132 - accuracy: 0.9962 - val_loss: 0.0358 - val_accuracy: 0.9888
Accuracy: 0.9887999892234802
 Error: 1.1200010776519775
```

Ce qui est à observer c'est que le traitement a été plus rapide que pour le modèle précédent. Aussi, nous avons une erreur bien supérieur au modèle de classification, bien que la précision soit un peu meilleure.

Réessayons avec un modèle à deux couches cachées.

<u>Réseau de neurones convolutionnel avec Keras (2 couches cachées)</u>

Nous redéfinissons notre modèle de convolution afin qu'il ait deux couches de convolution et de regroupement (MaxPooling) au lieu d'une seule couche de chacune.

```python
def convolutional_model():
 # create model
 model = Sequential()
 model.add(Conv2D(16, (5, 5), activation='relu', input_shape=(28, 28, 1)))
 model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
 model.add(Conv2D(8, (2, 2), activation='relu'))
 model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
 model.add(Flatten())
 model.add(Dense(100, activation='relu'))
 model.add(Dense(num_classes, activation='softmax'))
 # Compile model
 model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
 return model
```

Nous appelons maintenant la fonction pour créer notre nouveau réseau de neurones à convolution, puis nous entraînons et évaluons notre nouveau modèle.

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 14s - loss: 0.4370 - accuracy: 0.8771 - val_loss: 0.1107 - val_accuracy: 0.9662
Epoch 2/10
 - 12s - loss: 0.0983 - accuracy: 0.9702 - val_loss: 0.0706 - val_accuracy: 0.9781
Epoch 3/10
 - 12s - loss: 0.0709 - accuracy: 0.9786 - val_loss: 0.0641 - val_accuracy: 0.9801
Epoch 4/10
 - 12s - loss: 0.0572 - accuracy: 0.9823 - val_loss: 0.0539 - val_accuracy: 0.9830
Epoch 5/10
 - 12s - loss: 0.0513 - accuracy: 0.9843 - val_loss: 0.0440 - val_accuracy: 0.9869
Epoch 6/10
 - 12s - loss: 0.0444 - accuracy: 0.9871 - val_loss: 0.0457 - val_accuracy: 0.9852
Epoch 7/10
 - 14s - loss: 0.0393 - accuracy: 0.9880 - val_loss: 0.0420 - val_accuracy: 0.9858
Epoch 8/10
 - 13s - loss: 0.0359 - accuracy: 0.9894 - val_loss: 0.0389 - val_accuracy: 0.9878
Epoch 9/10
 - 11s - loss: 0.0326 - accuracy: 0.9905 - val_loss: 0.0389 - val_accuracy: 0.9877
Epoch 10/10
 - 11s - loss: 0.0306 - accuracy: 0.9908 - val_loss: 0.0343 - val_accuracy: 0.9895
Accuracy: 0.9894999861717224
 Error: 1.0500013828277588
```

Notre erreur est encore faible mais la précision a augmenté pour une durée de traitement à peu près similaire.

<u>Travail sur d'autres jeu de donnée</u>

Utilisons le modèle convolutionnel avec le jeu de données FASHION-MNIST précédemment étudié lors de l'étude du modèle classification de la partie juste avant .

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 12s - loss: 0.8050 - accuracy: 0.7173 - val_loss: 0.5628 - val_accuracy: 0.7840
Epoch 2/10
 - 13s - loss: 0.5044 - accuracy: 0.8152 - val_loss: 0.4911 - val_accuracy: 0.8201
Epoch 3/10
 - 12s - loss: 0.4542 - accuracy: 0.8354 - val_loss: 0.4557 - val_accuracy: 0.8364
Epoch 4/10
 - 12s - loss: 0.4264 - accuracy: 0.8468 - val_loss: 0.4312 - val_accuracy: 0.8472
Epoch 5/10
 - 13s - loss: 0.4033 - accuracy: 0.8546 - val_loss: 0.4237 - val_accuracy: 0.8476
Epoch 6/10
 - 12s - loss: 0.3866 - accuracy: 0.8622 - val_loss: 0.4044 - val_accuracy: 0.8573
Epoch 7/10
 - 14s - loss: 0.3709 - accuracy: 0.8675 - val_loss: 0.3933 - val_accuracy: 0.8606
Epoch 8/10
 - 13s - loss: 0.3558 - accuracy: 0.8715 - val_loss: 0.3764 - val_accuracy: 0.8668
Epoch 9/10
 - 12s - loss: 0.3457 - accuracy: 0.8765 - val_loss: 0.3767 - val_accuracy: 0.8649
Epoch 10/10
 - 12s - loss: 0.3324 - accuracy: 0.8803 - val_loss: 0.3528 - val_accuracy: 0.8764
Accuracy: 0.8763999938964844
 Error: 12.360000610351562
```

En suivant le même processus, nous obtenons des résultats similaires au modèle de classification. En effet, bien que l'erreur soit supérieure au 1er modèle, nous pouvons noter que la durée de traitement est nettement plus courte avec une plus grande précision là encore.