Université Paris Cité

Deep Learning

Prédiction des données MNIST avec uniquement 100 labels

Rose-Milca CENAT

Master II Machine Learning pour la Science des Données

Professeur : Blaise Hanczar

Utilisation de plusieurs méthodes pour la prédiction des données MNIST avec uniquement 100 labels

Introduction

Ce projet est un projet de Deep Learning, pour la deuxième année de master Machine Learning pour la Science des Données à l'université Paris Cité pour l'année académique 2023-2024. Il est basé sur la prédiction des données MNIST avec uniquement 100 labels. L'objectif est de bien comprendre l'architecture de différents réseaux de neurones et l'utilisation de plusieurs méthodes afin de les comparer.

Ce document contient la description des méthodes utilisées, la baseline : réseaux sur uniquement 100 exemples, les résultats obtenus, les références, le code en annexes.

Exploration des données

On commence par explorer les données du data set MNIST. On charge le jeu de données on utilise Matplotlib pour faire l'affichage de quelques images et de leurs labels.

Methode I

La première méthode que nous avons explorée est un Deep Neural Network et Pseudo-Label pour la suite, on fera du semi-supervisé.

Un Deep Neural Network, ou réseau de neurones profond, se distingue par une particularité : il est composé d'au moins deux couches. Ceci lui permet de traiter les données de manière complexe, en employant des modèles mathématiques avancés. En général, un Deep Neural Network a une couche d'entrée, une couche de sortie et au moins une couche entre les deux. Plus le nombre de couches est élevé, plus un réseau est dit « profond »[1].

On a utilisé un modèle qui possède une couche d'entrée, une couche de sortie et trois couches cachés. Les couches cachées ont pour fonction d'activation ReLU, et la couche de sortie Sigmoid. Ces choix ont été faits sur l'adaptation de l'article utilisé et de l'entrainement du modèle.

Bien avant d'utiliser nos données, nous les avons segmentés, en prenant 100 (x,y) et 59 900(x). Ensuite nous avons utilisé la fonction reshape afin de convertir les images en vecteur de taille 784, normaliser les valeurs entre 0 et 1 ensuite, et les classes en format one-hot à l'aide de la fonction to_categorical.

Apres que les données soient traitées nous avons utilisés les 100(x,y) pour faire l'apprentissage : compile et fit.

Pour compiler le modèle nous avons utilisé Adam comme optimiseur, categorical_crossentropy comme loss et le métrique accuracy. Le modèle est entrainé sur 10 epochs.

Nous avons obtenus :

loss : 0.39

accuracy : 0.95

Ensuite nous avons évalué le modèle avec la méthode evaluate et obtenu :

loss : 1.13

accuracy : 0.62

Nous avons utilisé des images de format png(pas du jeu de données, mais des images png semblables aux MNIST) pour faire la prédiction, peu de classes ont été bien prédites.

Ensuite nous avons entamés avec Pseudo Label.

On a utilisé le précédent modèle pour prédire les données non labélisés. Ensuite on sélectionne l'indice de la classe ayant la probabilité maximale, ce qui nous donne les pseudo-labels. Ensuite, ils sont encodés en format one-hot. Les données labélisées sont ensuite concaténées avec les pseudo-labels pour un autre entrainement avec les mêmes hyper paramètres. Sauf qu'on a utilisé une seule epoch.

On a obtenu :

loss : 0.25

accuracy : 0.90

On a ensuite évalué le model et obtenu :

loss : 2.82

accuracy : 0.62

Méthode II

La deuxième méthode est similaire à la première, on a juste ajouté le Drop Out avec un coefficient de 0.5. On a repris les mêmes étapes que dans la première méthode. Et cette fois, avec l'entrainement des données labélisées, on a obtenu :

loss : 2.098

accuracy : 0.23

Apres évaluation :

loss : 2.11

accuracy : 0.40

Avec l'entrainement sur les pseudo-labels et les données labélisées combinés ensemble, on a obtenu :

loss : 0.75

accuracy : 0.74

Et pour l'évaluation du modèle, on a obtenu :

loss : 5.13

accuracy : 0.42

Méthode III

La méthode III est l'utilisation de Denoising Auto Encodeur plus DNN plus pseudo label.

Le DAE a été utilisé sur 100 données mais seules les images ont été utilisées.

Apres la sélection des 100 labels, on les a bruités. Et le modèle a été entrainé sur ces données bruitées et avec epoch =10.

Ensuite on a implémenté un DNN ayant la même architecture que précédemment.

Pour la prochaine étape, le DAE a été utilisé pour prédire les données non labélisés (59 900(x)).

Et de ces données prédites on a pris le maximum des probabilités, convertit en one-hot encoding qui deviennent les pseudo-labels qui ont été combinés avec les données labélisés et comme précédemment le modèle DNN a été entrainé et on a obtenu,

loss : 0.025

accuracy : 0.99

Et après évaluation, on a obtenu :

loss : 13.28

accuracy : 0.09

Baseline (réseaux sur uniquement 100 exemples)

Implémentation du modèle DNN

```
[ ]  random.seed(42)
     (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
     # Aplatir et normaliser
     x_train = x_train.reshape(x_train.shape[0], -1) / 255.0
     x_test = x_test.reshape(x_test.shape[0], -1) / 255.0
```

```
[ ]  y_train = to_categorical(y_train, 10)
     y_test = to_categorical(y_test, 10)
```

```
[ ]  # Selection de 100 examples labelisés
     labeled_indices = np.random.choice(len(x_train), size=100, replace=False)
     x_labeled = x_train[labeled_indices]
     y_labeled = y_train[labeled_indices]
```

```
[ ]  model = tf.keras.models.Sequential([
         tf.keras.layers.Flatten(input_shape=(784,)),
         tf.keras.layers.Dense(128, activation='relu'),
         tf.keras.layers.Dense(128, activation='relu'),
         tf.keras.layers.Dense(128, activation='relu'),
         tf.keras.layers.Dense(10, activation='sigmoid')
     ])

     model.summary()
```

```
Model: "sequential_45"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_45 (Flatten)        (None, 784)               0

 dense_218 (Dense)           (None, 128)               100480

 dense_219 (Dense)           (None, 128)               16512

 dense_220 (Dense)           (None, 128)               16512

 dense_221 (Dense)           (None, 10)                1290

=================================================================
Total params: 134794 (526.54 KB)
Trainable params: 134794 (526.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

https://khayyam.developpez.com/articles/intelligence-artificielle/tensorflow/

```
[ ] model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[ ] model.fit(x_labeled, y_labeled, epochs=10)
```

```
Epoch 1/10
4/4 [==============================] - 1s 5ms/step - loss: 2.3126 - accuracy: 0.1200
Epoch 2/10
4/4 [==============================] - 0s 5ms/step - loss: 2.0695 - accuracy: 0.3800
Epoch 3/10
4/4 [==============================] - 0s 5ms/step - loss: 1.8709 - accuracy: 0.4400
Epoch 4/10
4/4 [==============================] - 0s 5ms/step - loss: 1.6831 - accuracy: 0.5700
Epoch 5/10
4/4 [==============================] - 0s 4ms/step - loss: 1.4551 - accuracy: 0.6600
Epoch 6/10
4/4 [==============================] - 0s 5ms/step - loss: 1.2209 - accuracy: 0.7400
Epoch 7/10
4/4 [==============================] - 0s 4ms/step - loss: 0.9804 - accuracy: 0.8100
Epoch 8/10
4/4 [==============================] - 0s 4ms/step - loss: 0.7636 - accuracy: 0.9100
Epoch 9/10
4/4 [==============================] - 0s 6ms/step - loss: 0.6057 - accuracy: 0.9200
Epoch 10/10
4/4 [==============================] - 0s 5ms/step - loss: 0.4634 - accuracy: 0.9500
```

```
[ ] test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)
```

```
313/313 - 1s - loss: 1.0311 - accuracy: 0.6933 - 645ms/epoch - 2ms/step
```

```
files = ["zero.png", "one.png", "two.png", "three.png", "four.png", "five.png", "six.png", "seven.png"]
for file in files:
    img = load_img(file, color_mode='grayscale')
    x = img_to_array(img)
    x = x.reshape(1, -1) / 255.0
    y = model.predict(x)
    predicted_class = np.argmax(y, axis=-1)

    print(file + " => " + str(predicted_class[0]) + "\n")
```

```
1/1 [==============================] - 0s 58ms/step
zero.png => 2

1/1 [==============================] - 0s 17ms/step
one.png => 7

1/1 [==============================] - 0s 17ms/step
two.png => 2
```

Références

Article: Pseudo-Label : The Simple and Efficient Semi-Supervised Learning

Method for Deep Neural Networks

https://pyimagesearch.com/2020/02/24/denoising-autoencoders-with-keras-tensorflow-and-deep-learning/

https://www.kaggle.com/code/manohar676/binary-classification-using-mlp-autoencoder

https://khayyam.developpez.com/articles/intelligence-artificielle/tensorflow/

https://medium.com/analytics-vidhya/implementing-a-deep-neural-network-from-scratch-8ed9988d2e4d

https://www.datacamp.com/tutorial/introduction-to-deep-neural-networks