Prétraitement des données

Avant d'introduire des données dans un réseau de neuronnes une étape de prétraitement est nécessaire.

Toutes les données et étiquettes introduites dans un réseau de neurones doivent être des tenseurs à virgule flottante.
Cette étape est la vectorisation. Les données doivent être dans un tableau numpy au format :' float32'

Exemple: train_image = train_image.astype('float32')

Les étiquettes de données doivent être vectorisées à l'aide d'un codage à chaud du type: to_categorical

Exemple: from tensorflow.keras.utils import to_categorical
train_labels = to_categorical(train_labels)

Nous pouvons aussi réorganiser les données d'entrée dans une nouvelle forme avec numpy:

Exemple:

Convertir l'image d'une matrice en vecteur:

image dans un tableau de forme (60000, 28, 28)
en :
train_images = train_images.reshape((60000, 28 * 28))

Les données d'images doivent être dans un tableau:

(échantillons, hauteur d'image, largeur d'image, nombre de canaux) avec:

image en niveau de gris ———nombre de canaux 1
image en couleur ——————nombre de canaux 3

Exemple:

caractéristique de taille (28, 28, 1) soit de taille 28 X 28 en 255 niveaux de gris.
caractéristique de taille (28, 28, 3) idem mais en couleurs.

Les données doivent être normalisées:

```
x = data
x -= x.mean(axis=0)
x /= x.std(axis=0)
```

avec sklearn:

```
x = data
scaler = StandardScaler()
```

x_normalized = scaler.fit_transform(x)

Fonction de perte et optimisateur.

Pendant la compilation du modele nous devons choisir une fonction de perte et un optimisateur.
La fonction de perte représente la quantité minimisée pendant l'entrainement.
L'optimisateur détermine comment mettre le réseau à jour en fonction de la fonction de perte.

Etape de compilation.

Fonction de perte ou de cout:
mean_squared_error ———— pour la régression
binary_crossentropy ———pour les prédictions d'étiquettes binaires
categorical_crossentropy —— pour la prédiction d'étiquettes multi-catégories

Les optimiseurs:

'sgd' — descente de gradient stochastique
'RMSprop' —— plus avancé
'adam' —— par défaut, le plus populaire

Les fonctions d'activation:

'Sigmoid' ou fonction logistique, c'est la plus utilisée. Sa sortie est centrée sur 0,5 avec une plage de 0 à 1
'Tanh' la sortie Tant est centrée sur 0 avec une plage de -1 à 1
'Relu' la Relu est la fonction d'activation la plus utilisée. Sa sortie n'a pas de valeur maximale.
'Softmax' elle est utile pour la classification multiclasse. Elle donne une probabilité.

Bonnes pratiques.

Nous pouvons utiliser un ensemble de données de validation pendant la formation pour voir dans quelle
mesure le modele se généralise.

Exemple:

Nous réservons 10% des données d'entrainement pour la validation avec ' validation_spilt'

history = mode.fit(x_train, y_train, validation_split = 0.1)

ou bien en utilisant un jeu de donnée de validation avec: 'validation_data'

history = model.fit(x_train, y_train, validation_data = (x_val, y_val)

Pour sauvegarder un modèle entrainé:

```
model.save(‘mon_model.h5)
```

pour le charger:

```
from tensorflow.keras.models import load_model
model = load_model(‘mon_model.h5)
```