TP1 - Prédiction de la langue d'un texte

masco programmation

11 février 2021

1 Objectif

L'objectif de ce projet est de programmer un perceptron multicouche permettant de prédire la langue d'un texte à partir de statistiques sur la fréquence des lettres dans ce texte.

2 Données

Les données permettant d'entraîner le modèle sont simples : du texte brut dans chacune des langues.

Les données sont divisées en *données d'apprentissage* qui vont servir à entraîner le modèle et en *données de test*, qui serviront à évaluer le modèle sur les données non vues lors de l'apprentissage.

3 Modèle

Le modèle utilisé dans ce projet est un perceptron multicouche, composé de deux couches : $W^{(1)}$ et $W^{(2)}$ et de trois niveaux, un niveau d'entrée ($\mathbf{x}$), un niveau caché ($\mathbf{a}$) et un niveau de sortie ($\mathbf{y}$).

La couche d'entrée contient les statistiques extraites du texte : la fréquence des lettres (unigrammes), la fréquence des séquences de deux lettres (bigrammes), ...Sa dimension D dépend de la nature des statistiques extraites.

Le niveau caché est composée de M neurones, chacun desquels est lié aux différents composants du niveau d'entrée. La valeur a_j du neurone j est calculée en multipliant la composante i du vecteur $\mathbf{x}$ par un poids $w_{j,i}^{(1)}$ puis en réalisant la somme de ces facteurs. Ce résultat est donné en entrée à une fonction d'activation h. En d'autre termes :

$$a_j = h\left(\sum_{i=1}^{n} w_{j,i}^{(1)} x_i + w_{j,0}^{(1)}\right) \tag{1}$$

Les *activations* a_j constituent elles-mêmes les entrées de la deuxième couche, dont les poids sont notés $w_{i,j}^{(2)}$ et la fonction d'activation h.

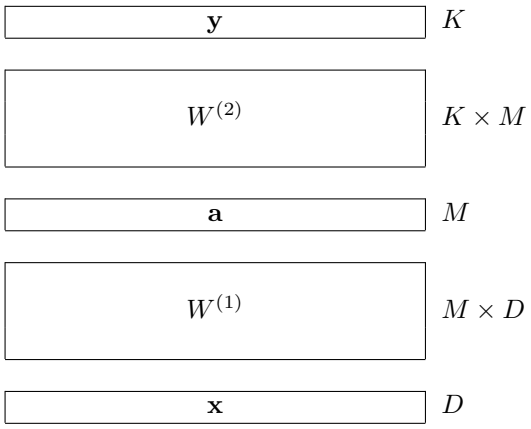

FIGURE 1 – Structure du réseau

$$y_k = h\left(\sum_{j=1}^{M} w_{k,j}^{(2)} a_j + w_{k,0}^{(2)}\right) \tag{2}$$

En remplaçant les termes a_j dans l'équation 3, par leur définition, donnée par l'équation 1, on obtient l'équation suivante :

$$y_k = h\left(\sum_{j=1}^{M} w_{k,j}^{(2)} h\left(\sum_{i=1}^{D} w_{j,i}^{(1)} x_i + w_{j,0}^{(1)}\right) + w_{k,0}^{(2)}\right) \tag{3}$$

4 Keras

Keras est une librairie python qui permet de construire des réseaux de neurones et d'en calculer les poids à partir de données. L'exemple suivant produit le modèle décrit dans la section précédente. La ligne 4 crée le modèle, qui est du type `sequential` : un empilement de couches. Les ligne 5 et 6 ajoutent successivement les deux couches. Elles sont du type `Dense`, qui indique que chaque neurone de la couche est relié à tous les neurones de la couche précédente. Le paramètre `units` indique la taille de la couche, `activation`, la fonction d'activation. La première couche indique de plus la dimension de l'entrée. La méthode `compile` spécifie la fonction de perte, ici l'entropie croisée ainsi que l'algorithme d'apprentissage, ici *Stochastic Gradient Descent*.

La méthode `fit` lance l'apprentissage du réseau, elle prend en paramètre les entrées et les sorties : `x_train` et `y_train`, le nombre d'itérations (`epochs`), la taille des *minibatches* et la part des données d'apprentissage qui ne servira pas pour l'apprentissage mais pour calculer l'évolution des performances du modèle aux différentes itérations.

La méthode `predict` effetue une prédiction. Elle prend en entrée un vecteur `x_test` et produit le vecteur `y_pred` qui contient un score pour chacune des K classes.

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(units=M, activation='relu', input_dim=D))
model.add(Dense(units=K, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=100, batch_size=1, validation_split=0.2)
y_pred = model.predict(x_test, batch_size=None, verbose=1, steps=None)
```

5 Numpy

La librairie `keras` utilise la librairie `numpy` qui permet de manipuler des tableaux multi-dimensionnels. En particulier, les deux variables `x_train` et `y_train` sont des tableaux `numpy`. Les tableaux `numpy` peuvent être créés à partir de structures `python` à l'aide de la fonction `array`. Cette dernière transforme des listes de listes en tableaux bi-dimensionnels, les listes de listes de listes en tableaux tri-dimensionnels et ainsi de suite, comme l'illustre l'exemple suivant :

```
>>> b = np.array([(1.5,2,3), (4,5,6)])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

6 Colab

Colab est une plateforme qui permet de de manipuler des réseaux de neurones programmés avec `keras`. Colab repose sur la notion de *notebook* qui sont des documents regroupant du texte et du code python. Un notebook est constitué de cellules, ces dernières pouvant être des cellules de texte ou de code.

Vous trouverez sur la page du cours le notebook `detecte_langue.ipynb` qui permet d'entraîner et d'évaluer un perceptron multicouche prédisant la langue d'un texte et qui vous servira comme base pour ce projet.

7 Ce qu'il faut faire

1. Ouvrir et exécuter le notebook `detecte_langue.ipynb`
2. Déterminer avec l'enseignant une étude à mener autour de ce réseau et de ces données (voir la section ci-dessous)
3. Réaliser l'étude sous la forme d'un notebook, à rendre pour le 21 février à minuit.

7.1 Quelques pistes à creuser

Voici quelques pistes possibles pour une étude à mener autour de ce réseau. Elles ne sont pas exclusives, vous pouvez choisir d'en explorer plusieurs :

- Taille des données d'apprentissage. Les performances obtenues par le classifieur dépendent du nombre d'exemples vus lors de l'apprentissage. Il est possible de tracer une courbe d'apprentissage donnant les performances du réseau selon la taille des données d'apprentissage pour chacune des langues. Toutes les langues ont-elles besoin d'autant de donnéés ? pourquoi ?
- Taille des données de test. Dans certains cas, on peut souhaiter que le classifieur fournisse une réponse le plus vite possible, cela veut dire en ayant vu qu'un tout petit échantillon du texte dont on veut identifier la langue. Il est possible de tracer une courbe indiquant les performances du classifieur en fonction de la taille des données de test. Si l'on souhaite que le classifieur ait une performance de $x\%$, on peut calculer, pour chaque langue, quelle est la taille des données de test nécessaires pour atteindre ce seuil. Est elle différente pour les différentes langues ?
- Structure du réseau et paramètres. Un perceptron multicouche comporte un grand nombre de paramètres : le nombre de couches cachées, la taille de ces dernières, les fonctions d'activation, la taille des minibatch, l'algorithme de d'apprentissage. Quels paramètres sont-ils importants dans notre cas ? Est ce qu'un modèle linéaire obtient de bons résultats ?
- Représentation en entrée. Dans l'implémentation fournie, le réseau prend en entrée les fréquences des bigrammes, est ce qu'un modèle prenant en compte les unigrammes suffit ? un modèle trigramme obtient-il de meilleures performances ? peut être que la fréquence de certaines lettres ou bigrammes ou trigramme suffit, comment faire pour le savoir ?
- On pourrait imaginer que le texte dont on cherche à déterminer la langue soit issu de la reconnaissance optique de caractères (OCR), auquel cas il peut comporter des erreurs (lettres confondues). On pourrait introduire du bruit dans nos données, en modifiant de manière aléatoire certaines lettres. Comment évoluent les performances avec le bruit ? vaut il mieux faire l'apprentissage du modèle avec des données bruitées ou des données propres ? Le modèle de bruitage décrit est un peu simpliste dans la mesure où il ne reproduit pas les erreurs véritablement effectuées par les système d'OCR. Comment faire un modèle plus réaliste ?
- Certaines langues sont elles plus difficiles à distinguer que d'autres ? est ce qu'une matrice de confusion nous permet de retrouver des familles de langues ?