TP3 OUX

March 16, 2023

1 TP3 : Exemple d'apprentissage du OUX (XOR) avec TensorFlow

Le tableau du OU Exclusif sans et avec Bias est :

a	b	a oux b	le tableau avec bias ->	bias	a	b	a oux b
0	0	0		1	0	0	0
0	1	1		1	0	1	1
1	0	1		1	1	0	1
1	1	0		1	1	1	0

Théoriquement, en 1 couche, l'apprentissage du OUX par réseau de neurones est alors impossible.

Importer les librairires

[1]:
```
#keras : Python Deep Learning library
import tensorflow.keras as keras
#prevision d'utiliser un réseau en couches séquentielles
from tensorflow.keras.models import Sequential
#prevision d'utiliser des couches totalement connectées la précédente
from tensorflow.keras.layers import Dense
#utilisation de la classique librairie pour tableaux, ...
import numpy as np
```

1.0.1 Définir les entrées et sorties attendues

- a OUX b est vraie seulement si un seule des deux variable a ou b est vraie

[2]:
```
# a et b sont les seules entrées
entrees = np.array([[0,0],[0,1],[1,0],[1,1]])

# une seule sortie
sorties = np.array([[0],[1],[1],[0]])
```

1.1 1. test du OUX - SANS COUCHE CACHEE

1.1.1 1.1. Choisir le modèle de réseau

- ici les couches sont séquentielles

[3]:
```
model = Sequential()
```

1.1.2 1.2. Définir l'architecture du réseau

- ici une seule couche constituée de 1 neurone en sortie,
- de 3 neurones en entrée (1 pour chaque valeur, +1 pour le Bias),
- utilisation de la sigmoïde comme fonction d'activation

[4]:
```
model.add(Dense(1, input_dim=2, use_bias=True, activation='sigmoid'))
```

1.1.3 1.3. Compiler le réseau

Ici, on précise que - l'algo de correction d'erreur est 'Adamax', - l'erreur calculée est la moyenne des valeurs absolues des erreurs commises

[5]:
```
model.compile(optimizer='adamax', loss='MSE')
```

1.1.4 1.4. Entraîner le réseau

- et on lance 10000 cycles d'apprentissage

[6]:
```
model.fit(entrees, sorties, verbose=0, epochs=1000)
```

[6]:
```
<keras.callbacks.History at 0x26e8ced3dc0>
```

1.1.5 1.5. Vérifier le réseau

Etape facultative, en général ***on teste le réseau sur d'autres exemples***. - Ici, on n'en a pas. Alors on lui demande de calculer la sortie pour chaque exemple de l'ensemble d'entraînement

[7]:
```
predictions = model.predict(entrees)
```

1.1.6 1.6. Affichage des résultats

Ici pas de nécessité de graphique d'évolution de l'erreur. On affiche les entrées, la sortie attendue, la sortie calculée.. et l'erreur..

[8]:
```
def verification(bias=False):
    print("verification")
    for i in range(0, len(entrees)):
        print(entrees[i][0], " - ", entrees[i][1], " attendu ", sorties[i], "␣
 ↪trouvé ",  predictions[i])

verification()
```

```
loss = model.evaluate(entrees, sorties,verbose=0)
print("perte=",loss)
```

```
verification
0  -  0  attendu  [0]  trouvé  [0.45923457]
0  -  1  attendu  [1]  trouvé  [0.5287289]
1  -  0  attendu  [1]  trouvé  [0.46306303]
1  -  1  attendu  [0]  trouvé  [0.53256613]
perte= 0.2512302100658417
```

Un beau "plantage", comme cela était attendu

Ajoutons maintenant une couche cachée..

1.2 2. test du OUX - AVEC COUCHE CACHEE

1.2.1 2.1. Choisir le modèle de réseau

- ici les couches sont séquentielles

[9]:
```
model = Sequential()
```

1.2.2 2.2. Définir l'architecture du réseau***

- une première couche composée de
  - 3 neurones en entrée : 2 neurones pour les 2 variables, plus le neurone BIAS
  - **et maintenant 4 neurones en sortie !!**
- une **couche intermédiaire**
  - *implicitement de 4 neurones en entrée* (ceux de la couche précédente) et
  - de 1 neurone en sortie (a OUX b)

[10]:
```
#une premiere couche constituée de 2 neurones en sortie, de 3 neurones en␣
↪entrée (2 pour les valeurs + 1 Bias), fonction d'activation = tangente␣
↪hyperbolique
model.add(Dense(4, input_dim=2, use_bias=True, activation='tanh'))

#une seconde couche constituée de 1 neurone en sortie (et implicitement de 2␣
↪neurones en entrée), fonction d'activation = tangente hyperbolique
model.add(Dense(1, activation='tanh'))
```

1.2.3 2.3. Compiler le réseau

- ici, on précise que l'algo de correction d'erreur est *'adam'*, et que l'erreur calculée est la moyenne des valeurs absolues des erreurs commises

[11]:
```
model.compile(optimizer='adam', loss='MSE')
```

1.2.4 2.4. Entraîner le réseau

- ici on ne le fait pas 'parler' (verbose=0), et on "ne lance plus que" 2000 cycles d'apprentissage

[12]:
```
model.fit(entrees, sorties, verbose=0, epochs=2000)
```

[12]:
```
<tensorflow.python.keras.callbacks.History at 0x6404334d0>
```

1.2.5 2.5. Vérifier le réseau

- Etape nécessaire; en général on le teste sur d'autres exemples. Ici, on lui demande de calculer la sortie pour chaque exemple de l'ensemble d'entraînement

[13]:
```
predictions = model.predict(entrees)
```

1.2.6 2.7. Affichage des résultats

- pas de courbe d'erreurs ici, on se contente d'afficher les entrées, la sortie attendue et la sortie calculées; et on affiche l'erreur

[14]:
```
verification()

loss = model.evaluate(entrees, sorties,verbose=0)
print("perte=",loss)
```

```
verification
0.0  -  0.0  attendu  [0.]  trouvé  [0.01189649]
0.0  -  1.0  attendu  [1.]  trouvé  [0.91050607]
1.0  -  0.0  attendu  [1.]  trouvé  [0.86376584]
1.0  -  1.0  attendu  [0.]  trouvé  [0.02558656]
perte= 0.006841277703642845
```

1.3 Importance de la couche intermédiaire validée !!!