# TP2 Et

March 16, 2023

# **TP2** : Exemple d'apprentissage du ET avec TensorFlow

## 0.1 Illustration de l'importance du Bias

| a | b | a et b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

*Théoriquement, en 1 couche, l'apprentissage du ET par réseau de neurones n'est pas possible.*

---

**Importer les librairies**

[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
```

---

## 0.2 Définir les entrées et sorties attendues

[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],[0],[0],[1]])
```

[3]:
```
sorties
```

[3]:
```
array([[0],
       [0],
```

```
[0],
[1]])
```

---

## 0.3 1. Version sans BIAS

### 0.3.1 1.1. Choisir le modèle de réseau

***ici les couches sont séquentielles***

[4]:
```
model = Sequential()
```

### 0.3.2 1.2. Définir l'architecture du réseau

- ici une seule couche constituée de 1 neurone en sortie,
- de 2 neurones en entrée (pour chaque valeur),
- utilisation de la sigmoïde comme fonction d'activation

[5]:
```
model.add(Dense(1, input_dim=2, use_bias=False, activation='sigmoid'))
```

---

### 0.3.3 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 MSE.

[6]:
```
model.compile(optimizer='adamax', loss='MSE')
```

---

### 0.3.4 1.4. Entraîner le réseau

- ici on ne le fait pas parler (verbose = 0),
- et on lance 10000 cycles d'apprentissage

[7]:
```
model.fit(entrees, sorties, verbose=0, epochs=10000)
```

[7]:
```
<keras.callbacks.History at 0x24314a88e80>
```

---

### 0.3.5 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

[8]:
```
predictions = model.predict(entrees)
```

[9]:
```
predictions
```

[9]:
```
array([[0.5       ],
       [0.49999973],
       [0.49999973],
       [0.49999946]], dtype=float32)
```

[10]:
```
ws = model.get_weights()
ws
```

[10]:
```
[array([[-1.1042832e-06],
        [-1.0214852e-06]], dtype=float32)]
```

---

### 0.3.6 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 ainsi que les poids appliquées aux entrées et au signal bias..

[11]:
```
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])

    ws = model.get_weights()
    print("poids pour entree x = " + str(ws[0][0][0]))
    print("poids pour entree y = " + str(ws[0][1][0]))
    if(bias):print("poids pour bias = " + str(ws[1][0]))

verification()
```

```
verification
0  -  0  attendu  [0]  trouvé  [0.5]
0  -  1  attendu  [0]  trouvé  [0.49999973]
1  -  0  attendu  [0]  trouvé  [0.49999973]
1  -  1  attendu  [1]  trouvé  [0.49999946]
poids pour entree x = -1.1042832e-06
poids pour entree y = -1.0214852e-06
```

[12]:
```
loss = model.evaluate(entrees, sorties,verbose=0)
print("perte=",loss)
```

```
perte= 0.25
```

**Des erreurs importantes donc**, comme prévu....

---

## 0.4 2. Version AVEC BIAS

Le tableau est alors

| bias | a | b | a et b |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

*Théoriquement, en 1 couche, l'apprentissage du ET par réseau de neurones est alors possible.*

### 0.4.1 2.1. Définir l'architecture du réseau

- ici une seule couche constituée de 1 neurone en sortie,
- de 3 neurones en entrée (2 contenant les valeurs + **un Bias** (émettant toujours le signal 1)),
- utilisation de la sigmoide comme fonction d'activation

[29]:
```
model = Sequential()
model.add(Dense(1, input_dim=2, use_bias=True, activation='sigmoid'))
```

---

### 0.4.2 2.2 Compiler et entrainer 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 - 10000 cycles d'apprentissage demandés

[30]:
```
model.compile(optimizer='adamax', loss='MSE')

model.fit(entrees, sorties, verbose=0, epochs=12000)
```

[30]:
```
<keras.callbacks.History at 0x243177b5f40>
```

---

### 0.4.3 2.2. Vérifier le réseau

Pas d'exemples de validation, on vérifie simplement la correspondance entre sortie attendue et la sortie réelle.

[31]:
```
predictions = model.predict(entrees)
verification(True)
```

```
verification
0  -  0  attendu  [0]  trouvé  [5.801788e-06]
0  -  1  attendu  [0]  trouvé  [0.01760811]
1  -  0  attendu  [0]  trouvé  [0.01760533]
1  -  1  attendu  [1]  trouvé  [0.98225796]
poids pour entree x = 8.035553
poids pour entree y = 8.035713
poids pour bias = -12.057344
```

[32]:
```
loss = model.evaluate(entrees, sorties, verbose=0)
print("perte=",loss)
```

```
perte= 0.00023369328118860722
```