



---

# *Machine Learning*

PROJECT : IMAGE CLASSIFICATION AND REGRESSION

---

Jérémy BEZAMAT

ENSEIRB-MATMECA

28 mai 2018

Enseignant : Vincent Lepetit

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Simple Classification</b>	<b>3</b>
<b>3</b>	<b>Problème de classification plus difficile</b>	<b>6</b>
<b>4</b>	<b>Problème de régression</b>	<b>8</b>

## 1 Introduction

Ce projet à lieu dans le cadre de l'option Machine Learning de l'ENSEIRB-MATMECA. Il a pour objectif d'appréhender quelques concepts du machine learning grâce à l'API python Keras. Le sujet de ce projet se trouve à l'adresse suivante : [http://www.labri.fr/perso/vlepetit/teaching/ml\\_dl\\_enseirb/project.pdf](http://www.labri.fr/perso/vlepetit/teaching/ml_dl_enseirb/project.pdf)

Le code est composé d'un premier fichier **mpl.py** fournit par l'enseignant qui permet de générer des datas et de visualiser des prédictions. Mais aussi d'un fichier **utile.py** qui contient une fonction *linear\_classifier* permettant de créer un classifieur linéaire, d'une fonction de test pour voir si le classifieur lineaire marche bien, d'une fonction de visualisation des différentes colonnes renvoyé par *model.get\_weights*, d'une fonction *deep\_network* permettant de créer un Deep Network composé d'une couche de convolution, d'une couche de pooling et d'une couche entièrement connectée et enfin d'une fonction de création d'un Deep Network adapté au problème de régression de la question 5. Le dernier fichier est **main.py** permet de lancer le programme.

Vous pouvez retrouver ce code à l'adresse suivante : [https://github.com/jbezamat/project\\_machine\\_learning\\_image\\_classification\\_and\\_regression\\_ENSEIRB-MATMECA](https://github.com/jbezamat/project_machine_learning_image_classification_and_regression_ENSEIRB-MATMECA)

## 2 Simple Classification

Dans cette partie, nous utilisons des données "simples" car avec une *free\_location* à False et un *noise* à 0.0.

Pour créer notre classifieur linéaire, nous avons créé la fonction *linear\_classifier* :

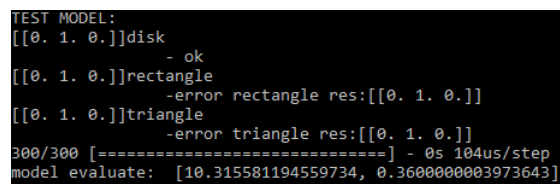
Listing 1 – *linear\_classifier*

```
1 def linear_classifier(X_train = None, Y_train= None, nb_data = 500,
2                       data_noise = 0.0, data_free_location = False,
3                       activation='softmax', loss='categorical_crossentropy',
4                       optimizer = 'adam', epochs = 15):
5     if X_train is None and Y_train is None:
6         [X_train, Y_train] = generate_dataset_classification(nb_data,
7                                                             data_noise, data_free_location)
8
9     model = Sequential()
10    model.add(Dense(3, input_shape=(generate_a_rectangle().shape[0],)))
11    model.add(Activation(activation))
12
13    if optimizer == 'sgd':
14        sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
15        model.compile(loss=loss, optimizer='sgd', metrics=['accuracy'])
16    else:
17        model.compile(loss=loss, optimizer=optimizer, metrics=['accuracy'])
18
19    model.fit(X_train, Y_train, epochs=epochs, batch_size=32)
20    return model
```

---

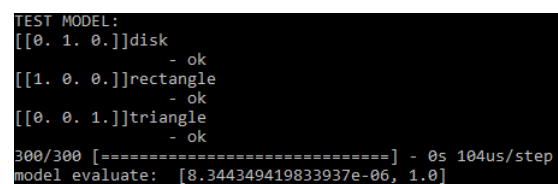
Cette fonction créée, grâce à l'API Keras, un modèle séquentiel (ligne 9), une couche qui prend en entrée la taille d'une image générée et en sortie 3 valeurs (ligne 10). Cette couche est ensuite activée, avec la fonction *softmax* par défaut (ligne 11). Puis optimisée et compilée (lignes 13 - 17). Finalement elle est entraînée avec les données [X\_train, Y\_train].

Nous avons ensuite testé ce classifieur linéaire en faisant varier plusieurs paramètres. Voilà par exemple les résultats en utilisant *SGD* ou *ADAM* comme optimisation avec 300 données d'entraînement, 300 de tests, et 1000 epochs :



```
TEST MODEL:
[[0. 1. 0.]]disk
- ok
[[0. 1. 0.]]rectangle
-error rectangle res:[[0. 1. 0.]]
[[0. 1. 0.]]triangle
-error triangle res:[[0. 1. 0.]]
300/300 [=====] - 0s 104us/step
model evaluate: [10.315581194559734, 0.3600000003973643]
```

FIGURE 1 – SGD



```
TEST MODEL:
[[0. 1. 0.]]disk
- ok
[[1. 0. 0.]]rectangle
- ok
[[0. 0. 1.]]triangle
- ok
300/300 [=====] - 0s 104us/step
model evaluate: [8.344349419833937e-06, 1.0]
```

FIGURE 2 – ADAM

La première valeur de 'model evaluate' correspond à la Loss fonction et la deuxième à l'accuracy. On voit bien que nous obtenons de meilleurs résultats avec *ADAM*

Une autre exemple ici avec les résultats en utilisant 1000 données, 50 000 epochs et *ADAM* qui montre que plus on augmente les epochs avec *ADAM* et plus les résultats semblent cohérents :

```
TEST MODEL:
[[0. 1. 0.]]disk
- ok
[[1. 0. 0.]]rectangle
- ok
[[0. 0. 1.]]triangle
- ok
300/300 [=====] - 0s 156us/step
model evaluate: [1.1920928955078125e-07, 1.0]
```

FIGURE 3 – ADAM 1000 données 500000 epochs

Nous avons également tracé les différentes courbes de poids du classifieur grâce à la fonction *visualize\_column*

Listing 2 – *visualize\_column*

---

```
1 def visualize_column(model):
2     c1 = model.get_weights()[0][:,0]
3     plt.imshow(c1.reshape(100,100), cmap='gray')
4     plt.show()
5
6     c2 = model.get_weights()[0][:,1]
7     plt.imshow(c2.reshape(100,100), cmap='gray')
8     plt.show()
9
10    c3 = model.get_weights()[0][:,2]
11    plt.imshow(c3.reshape(100,100), cmap='gray')
12    plt.show()
```

---

Voilà les images obtenues pour 1000 données, 50 000 epochs et *ADAM* :

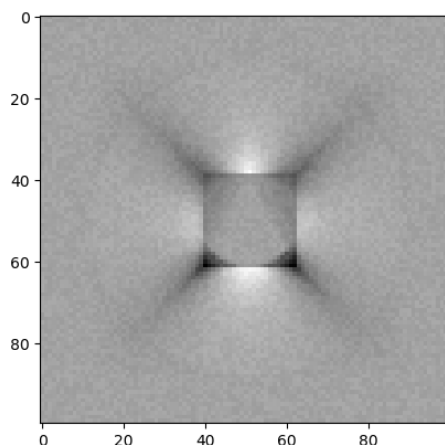


FIGURE 4 – Rectangle colonne 0

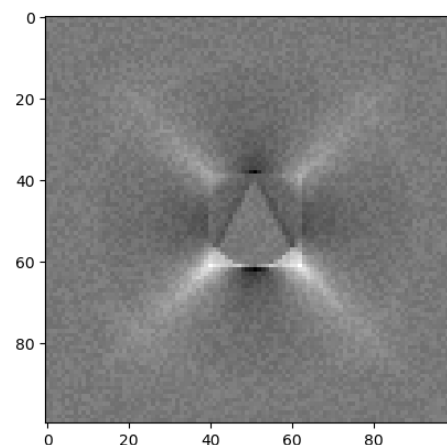


FIGURE 5 – Disque colonne 1

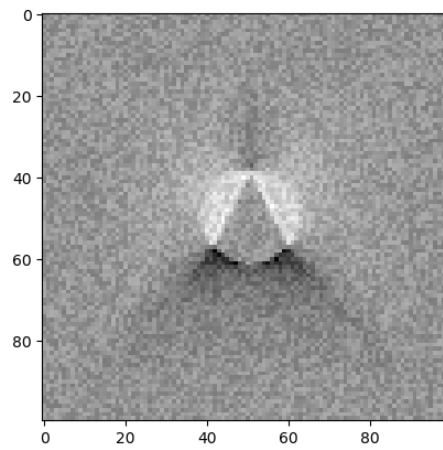


FIGURE 6 – Triangle colonne 1

### 3 Problème de classification plus difficile

Dans cette partie nous utilisons des données plus compliquées que la partie précédente car avec une *free\_location* à True et un *noise* à 20. De plus, nous avons utilisé ici 400 données d'entraînement, 300 de tests, 200 epochs et l'optimiseur *ADAM*.

Dans un premier temps nous avons essayé d'appliquer notre classifieur linéaire sur ce nouveau problème pour pouvoir ensuite comparer les résultats avec le Deep Network :

```
model evaluate: [8.890076738993328, 0.36666666666666664]
```

FIGURE 7 – Classifieur linéaire

Nous avons créé un Deep Network composé d'une couche de convolution 16 5\*5 filters, d'une couche de pooling et d'une couche entièrement connectée grâce à la fonction *deep\_network* :

Listing 3 – *deep\_network*

---

```
1 def deep_network(X_train = None, Y_train= None, nb_data = 500,
2                 data_noise = 0.0, data_free_location = False,
3                 activation1 = 'relu', activation2 = 'softmax',
4                 loss = 'categorical_crossentropy',
5                 optimizer = 'adam', epochs = 15):
6
7
8     if X_train is None and Y_train is None:
9         [X_train, Y_train] = generate_dataset_classification(nb_data,
10                                                            data_noise, data_free_location)
11
12     X_train = X_train.reshape(X_train.shape[0], 100, 100, 1)
13     X_train = X_train.astype('float32')
14
15     model = Sequential()
16     model.add(Conv2D(16, (5,5), activation=activation1,
17                    input_shape=(100, 100, 1)))
18     model.add(MaxPooling2D(pool_size=(2,2)))
19     model.add(Flatten())
20     model.add(Dropout(0.5))
21     model.add(Dense(3, activation=activation2))
22     if optimizer == 'sgd':
23         sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
24         model.compile(loss=loss, optimizer='sgd', metrics=['accuracy'])
25     else:
26         model.compile(loss=loss, optimizer=optimizer, metrics=['accuracy'])
27
28     model.fit(X_train, Y_train, epochs=epochs, batch_size=32)
29
30     return model
```

---

Voilà les résultats obtenus :

```
model evaluate: [1.621263518333435, 0.6433333333333333]
```

FIGURE 8 – Deep Network

Les résultats sont meilleurs qu'avec le classifieur linéaire mais ne sont pas encore optimaux. Ils pourraient être améliorés en augmentant le nombre de données d'entraînement ou le nombre d'époques par exemple.



## 4 Problème de régression

Dans cette partie nous utilisons des données de regression avec une *free\_location* à True et un *noise* à 20. De plus, nous avons utilisé ici 400 données d'entraînement, 300 de tests, 200 epochs et l'optimiseur *ADAM*.

Il fallait dans cette partie adapté le Deep Network de la partie précédente car nous ne voulions plus 3 valeurs en sortie (différenciant triangles, rectangles et disques) comme précédemment mais 6 valeurs décrivant les coordonnées des sommets d'un triangle.

Nous avons pour cela seulement changer la ligne `model.add(Dense(3,activation=activation2))` (ligne 21) par `model.add(Dense(6))`.

Voilà les résultats obtenus :

```
model evaluate: [0.08087464481592178, 0.2500000001986821]
```

FIGURE 9 – Model Evaluate Regression

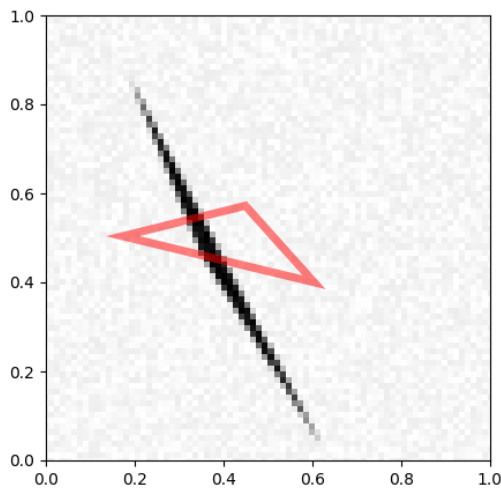


FIGURE 10 – visualize\_prediction  
X\_test[0]

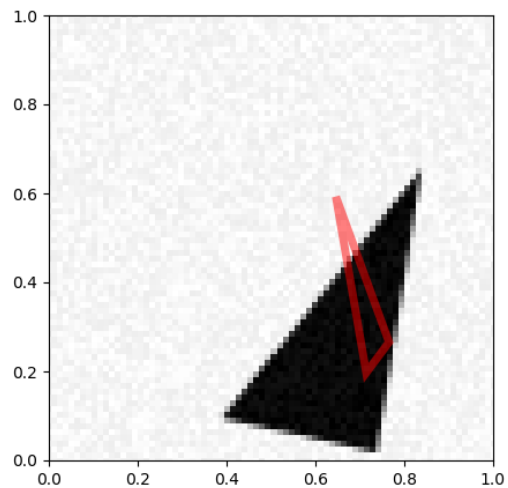


FIGURE 11 – visualize\_prediction  
X\_test[1]

Nous voyons bien que notre Deep Network n'est pas adapté à ce problème. En effet il ne sait approcher que des fonctions ce qui n'est pas le cas ici.