Projet et bureau d'études - Tensorflow - *A. Gibault*

Laboratoire 1 à 3

Jordan MEURANT

jordan.meurant@student.hepl.be

27 août 2022

Table des matières

1 Laboratoire 1 **3**
1.1 Exercice 1 - Opérateur logique ET . . . 3
1.1.1 Représentation du modèle obtenu et des données d'apprentissage . . . 3
1.1.2 Callback et fonction EarlyStopping . . . 3
1.1.3 Modèles obtenus . . . 4
1.1.4 SGD pour tensorflow, c'est vraiment stochastique ? . . . 5
1.2 Exercice 2 - Classification de données linéairement séparables . . . 6
1.3 Exercice 3 - Classification de données non-linéairement séparables . . . 8
1.4 Exercice 4 - Régression linéaire . . . 10
1.4.1 Prédictions par rapport à des échantillons tirés aléatoirement sur l'intervalle [-10,100] 11

2 Laboratoire 2 **12**
2.1 Classification de données . . . 12
2.2 Exercice 1 - Classification de données avec de nombreuses entrées . . . 12
2.3 Exercice 2 - Classification de données avec de nombreuses entrées . . . 13
2.3.1 Représentation des données . . . 13
2.3.2 Représentation des poids donnés à chacun des pixels pour chacun des symboles. . . . 14
2.3.3 Représentation du modèle et de l'évolution de la précision des prédictions . . . 15
2.4 Exercice 3 - Classification des Iris . . . 15
2.4.1 Représentation des données à classifier . . . 15
2.4.2 Entraînement avec 3 neurones . . . 17
2.5 Performances des différents modèles . . . 17

3 Laboratoire 3 **19**
3.1 Exercice 1 - Classification de données XOR . . . 19
3.2 Exercice 2 - Classification de données non-linéairement séparables . . . 20
3.2.1 Modèles . . . 20
3.2.2 Interprétation des zones de recouvrement et des zones vides . . . 21
3.2.3 Variations du nombre de neurones . . . 21
3.3 Exercice 3 - Classification des Iris . . . 21
3.3.1 Variations de 2 à 5 couches . . . 21

1 Laboratoire 1

1.1 Exercice 1 - Opérateur logique ET

1.1.1 Représentation du modèle obtenu et des données d'apprentissage

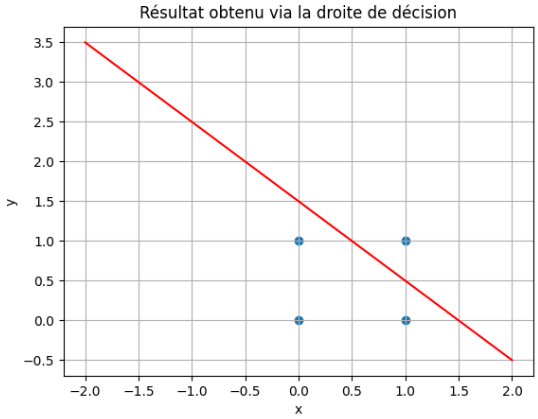

FIGURE 1 – Model obtenu et données d'apprentissage

1.1.2 Callback et fonction EarlyStopping

Lorsque nous construisons un model, le choix du nombres d'epochs peut paraître difficile. En effet, si celui-ci est élevé ou plutôt exagéré, notre modèle va "coller" nos données, va s'ajuster trop aux données d'apprentissage et ne se généralisera pas correctement. Ainsi, si on soumet celui-ci à un nouveau jeu de données, notre modèle pourrait mal-fonctionner et ainsi prédire des données aberrantes. Ce problème est appelé sur-ajustement ou overfitting en anglais.

Afin d'atténuer ce phénomène et obtenir un modèle qui se généralise correctement, une solution possible est d'obtenir un modèle avec un nombre d'epochs raisonnable.

Pour ce faire, dans le laboratoire, j'ai utilisé des callback : la fonction de callback **EarlyStopping()** et un callback personnalisé. Voici leurs avantages et inconvénients :

Avantages EarlyStopping

- Documentation complète.
- Au vu de sa documentation, il est rapide à mettre en place : il suffit d'appeler la fonction EarlyStopping dans le model.fit :

```
model.fit(..., callbacks=c.EarlyStopping(monitor="loss", mode="auto", min_delta
=0.002, patience=3, verbose=1))
```

- Permet d'arrêter l'apprentissage automatiquement lorsqu'il n'y a plus de changement sur la valeur surveillée. Effectivement, à l'aide de `min_delta` [1], on peut spécifier lorsque notre valeur à surveiller (ici la fonction coût) n'évolue plus et qu'on peut ainsi arrêter l'apprentissage.

1. la modification de la valeur a surveillé qui doit être plus grande à min_delta sous peine d'arrêter l'apprentissage.

Inconvénients EarlyStopping((

— Malgré son efficacité, le `min_delta` est l'inconvénient majeur de la fonction EarlyStopping. Le tout réside à trouver la bonne valeur. Effectivement si celle-ci est trop grande, l'apprentissage s'arrêtera après quelques epochs et on obtient un modèle absolument pas fiable. A l'opposé, si elle est trop petite, l'apprentissage risque de continuer vers le nombre d'epochs maximum ce qui n'arrange pas non plus.

— Pas customizable à souhait. Contrairement à un callback personnalisé, on ne peut pas lui spécifier un nombre, par exemple un seuil, où on aimerait arrêter l'apprentissage.

Avantages callback personnalisé

— Entièrement personnalisable : on peut le configurer à sa guise. Par exemple, dans le laboratoire, un seuil a été défini à `0.3`. Une fois atteint, on arrête l'apprentissage.

Inconvénients callback personnalisé

— Plus long à mettre en place comparé au **EarlyStopping()**. Effectivement, il demande à créer une classe, là où EarlyStopping est déjà prêt à l'emploi.

— Plus long à paramétrer : dans le cadre du seuil, il faut tester son modèle afin d'obtenir le meilleur seuil possible pour obtenir un modèle le plus généralisable possible.

1.1.3 Modèles obtenus

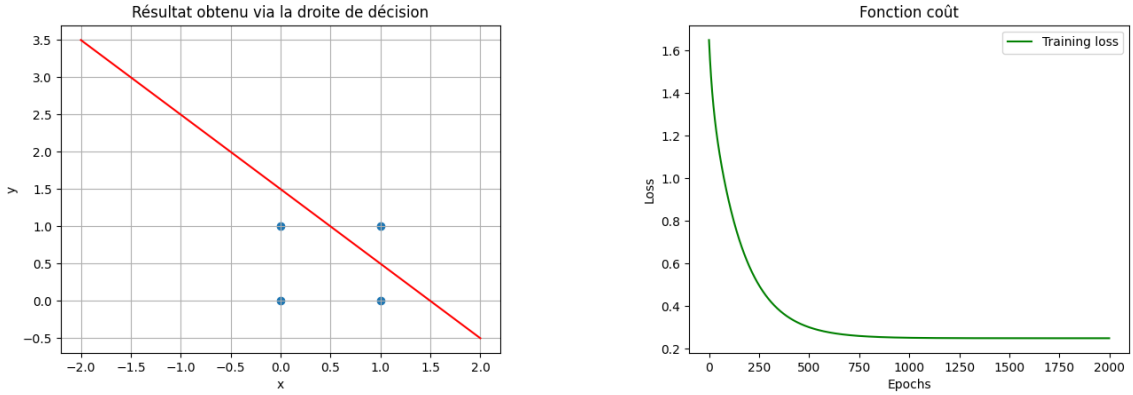

FIGURE 2 – modèle obtenu(lr=0.01,epochs=2000) - valeurs fonction coût et itérations avant arrêt

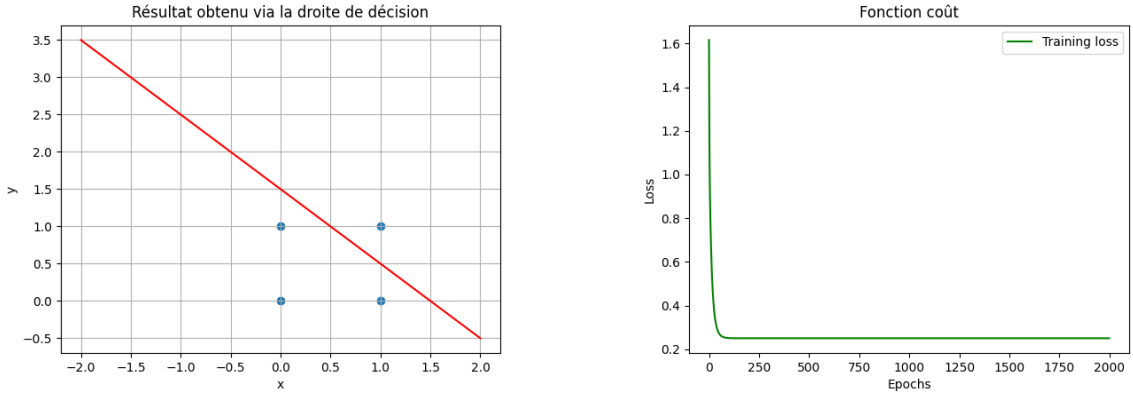

FIGURE 3 – modèle obtenu(lr=0.1,epochs=2000) - valeurs fonction coût et itérations avant arrêt

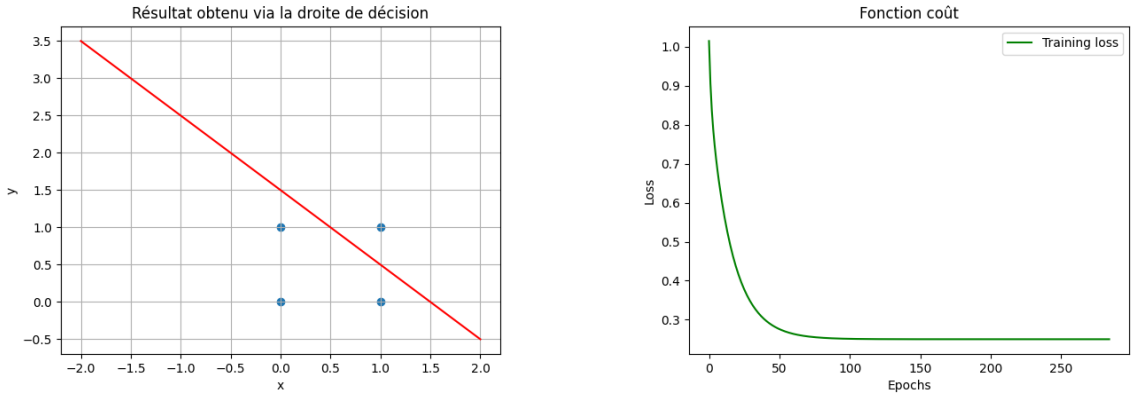

FIGURE 4 – modèle obtenu(lr=0.1,epochs=2000,seuil=0.26) - valeurs fonction coût et itérations avant arrêt(282)

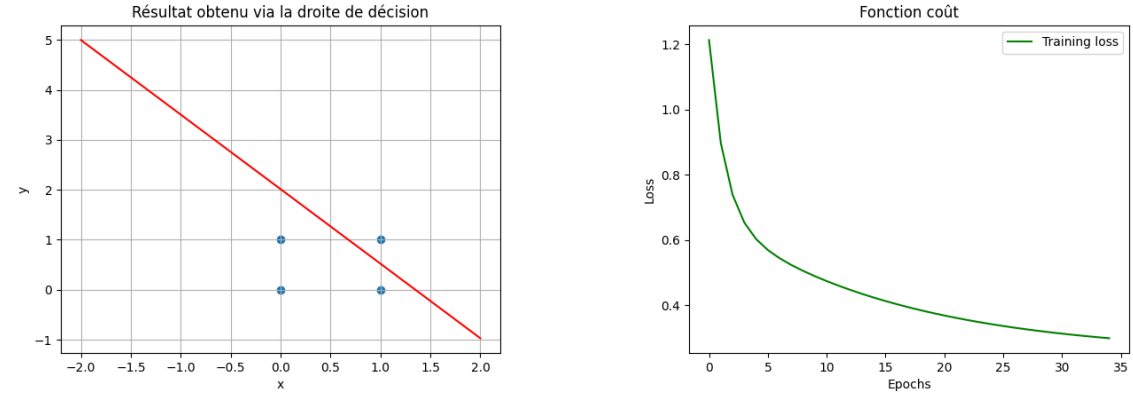

FIGURE 5 – modèle obtenu(lr=0.1,epochs=2000,seuil=0.3) - valeurs fonction coût et itérations avant arrêt(epochs=34)

Paramètres utilisés durant l'apprentissage : il s'est avéré qu'un learning rate de `0.1` permettait de converger plus rapidement. De plus, le seuil de `0.3`, est le seuil considéré le plus efficace afin d'obtenir un modèle avec de bonnes prédictions et avec le moins d'itérations possibles(34) contrairement au seuil de `0.26` qui pour lequel il aura fallu `282` itérations pour l'atteindre.

1.1.4 SGD pour tensorflow, c'est vraiment stochastique ?

Premièrement, que signifie stochastique ? Si on regarde une définition, on pourrait lire : "qui se produit par l'effet du hasard". Dans le cadre du machine learning, le sens du mot stochastique correspond à des données aléatoires qui seront évaluées par la technique d'apprentissage choisie(*Note : tous les techniques d'apprentissage ne tirent pas que des données aléatoires. C'est le cas du Gradient Descent qui considère l'ensemble des données à chaque étape.*

SGD - Stochastic Gradient Descent : il s'agit d'une technique d'apprentissage qui va calculer le gradient à partir d'un sous-ensemble **aléatoire** de ces données (ceci appelé en français mini-lot). Autrement dit, outre

son nom, il s'agit bien d'une technique d'apprentissage stochastique.

1.2 Exercice 2 - Classification de données linéairement séparables

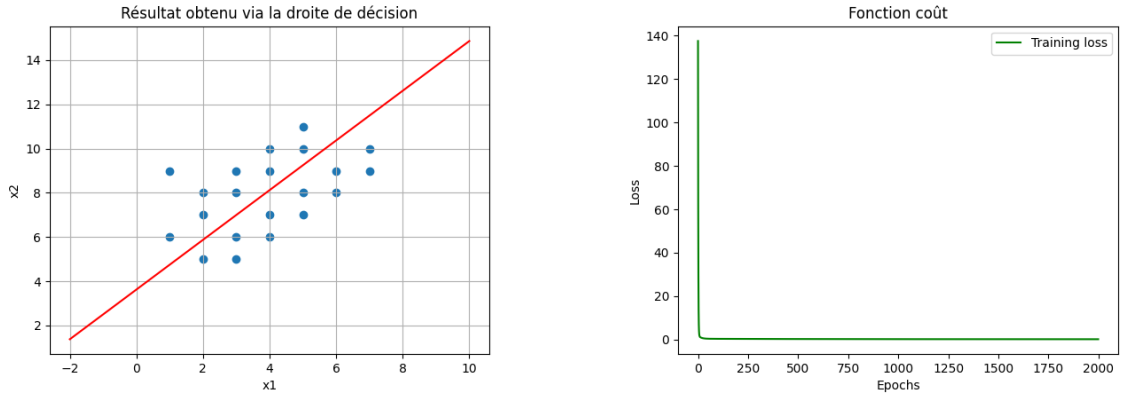

FIGURE 6 – modèle obtenu(lr=0.01,epochs=2000) - valeurs fonction coût et itérations avant arrêt(epochs=2000)

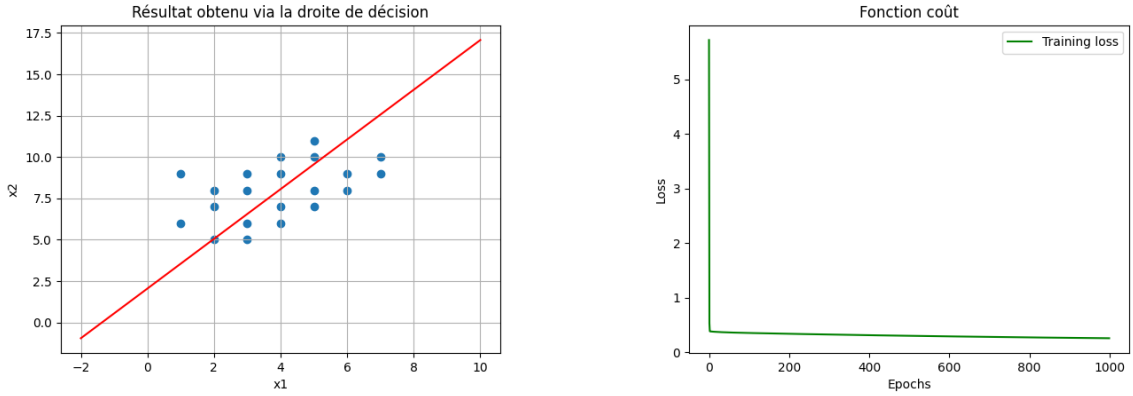

FIGURE 7 – modèle obtenu(lr=0.005,epochs=1000) - valeurs fonction coût et itérations avant arrêt(epochs=1000)

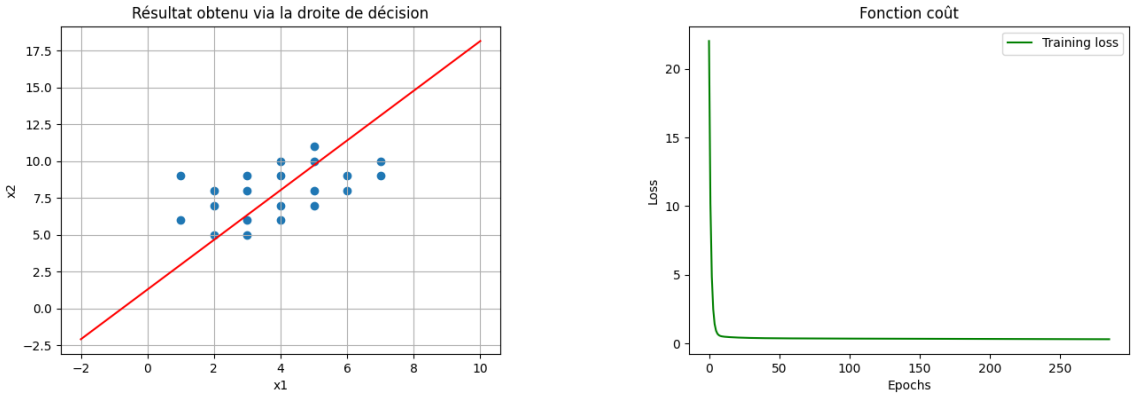

FIGURE 8 – modèle obtenu(lr=0.01,epochs=1000,seuil=0.3) - valeurs fonction coût et itérations avant arrêt(epochs=282)

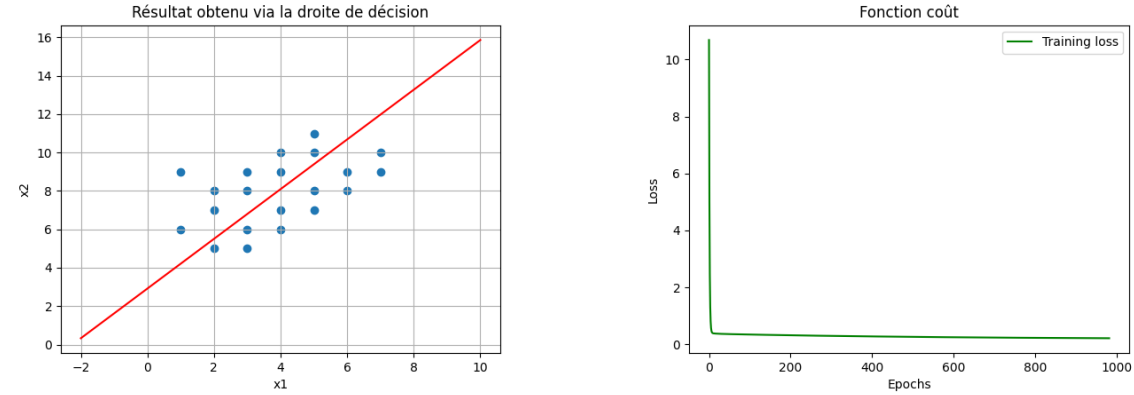

FIGURE 9 – modèle obtenu(lr=0.01,epochs=1000,seuil=0.21) - valeurs fonction coût et itérations avant arrêt(epochs=982)

Paramètres utilisés durant l'apprentissage : il s'est avéré qu'un learning rate de `0.1` permettait de converger plus rapidement. De plus, le seuil de `0.21`, est le seuil considéré le plus efficace afin d'obtenir un modèle avec de bonnes prédictions et avec le moins d'itérations possibles(982) contrairement au seuil de `0.24` qui effectue moins d'itérations(672) mais pour lequel notre modèle commence à décliner (Figure 10).

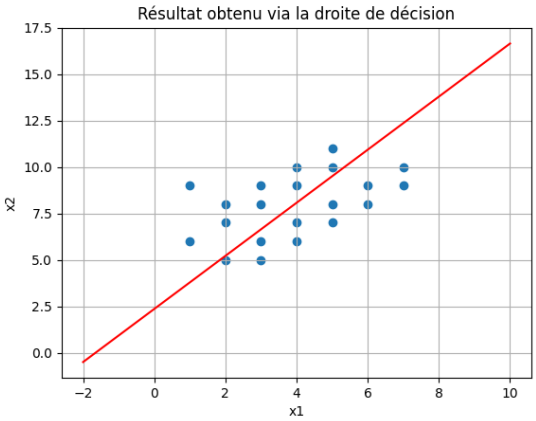

FIGURE 10 – modèle obtenu(lr=0.01,epochs=1000,seuil=0.24) - valeurs fonction coût et itérations avant arrêt(epochs=672)

1.3 Exercice 3 - Classification de données non-linéairement séparables

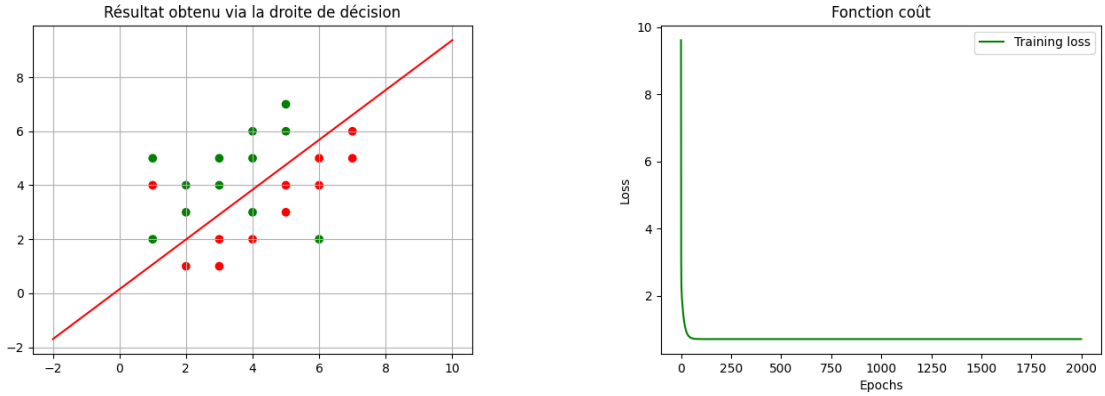

FIGURE 11 – modèle obtenu(lr=0.01,epochs=2000) - valeurs fonction coût et itérations avant arrêt

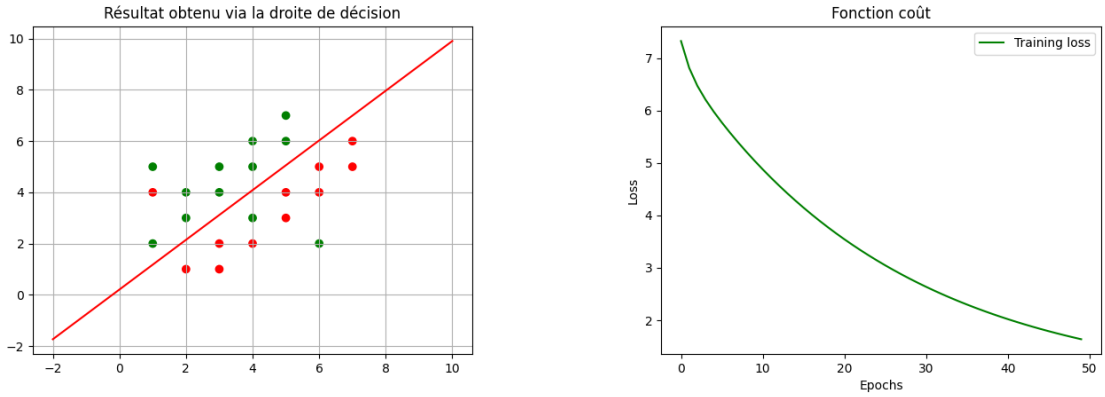

FIGURE 12 – modèle obtenu(lr=0.005,epochs=50) - valeurs fonction coût et itérations avant arrêt(loss=1.63,epochs=50)

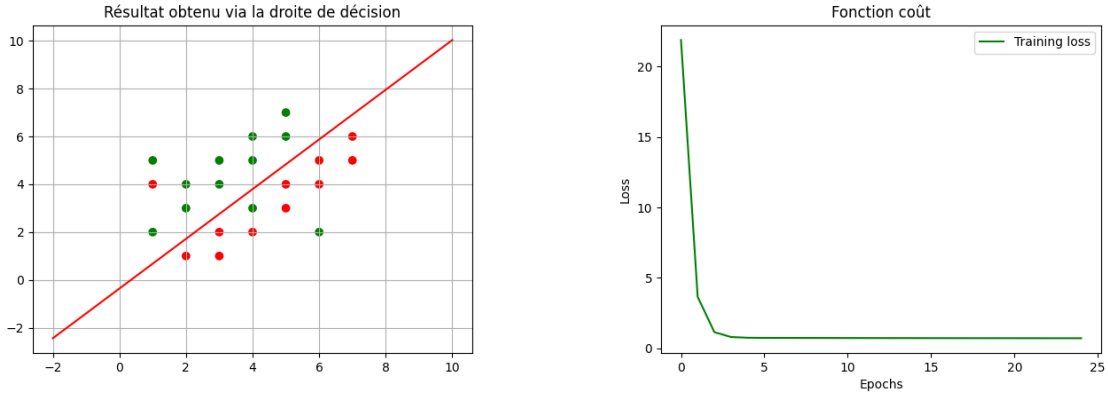

FIGURE 13 – modèle obtenu(lr=0.009,epochs=100,seuil=0.725) - valeurs fonction coût et itérations avant arrêt(epochs=25)

Paramètres utilisés durant l'apprentissage : il s'est avéré qu'un learning rate de `0.009` permettait de converger plus rapidement. De plus, le seuil de `0.725`, est le seuil considéré le plus efficace afin d'obtenir un modèle avec de bonnes prédictions et avec le moins d'itérations possibles : seulement 25. Effectivement, ce seuil peut paraître élevé mais il ne fait pas oublier qu'il s'agit de données non-linéairement séparables ce qui explique pourquoi le modèle comment des erreurs.

1.4 Exercice 4 - Régression linéaire

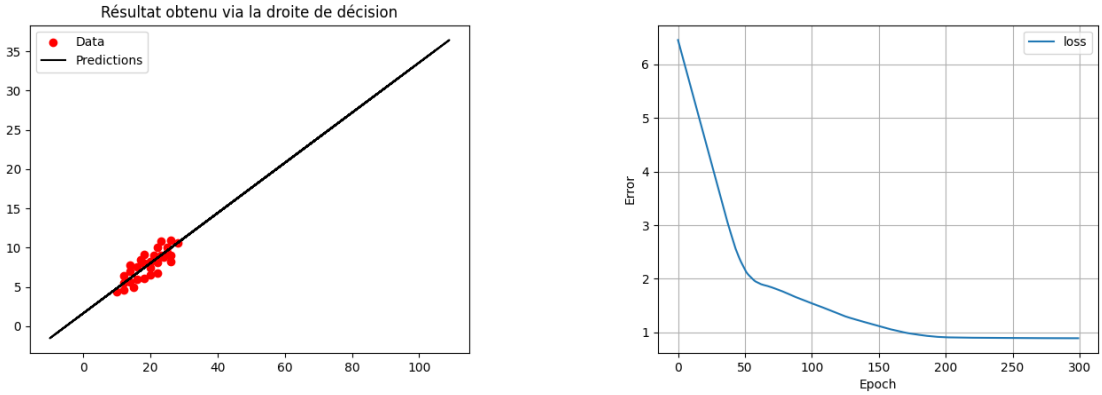

FIGURE 14 – modèle obtenu(lr=0.005,epochs=300) - valeurs fonction coût et itérations avant arrêt(loss=0.89,epochs=200)

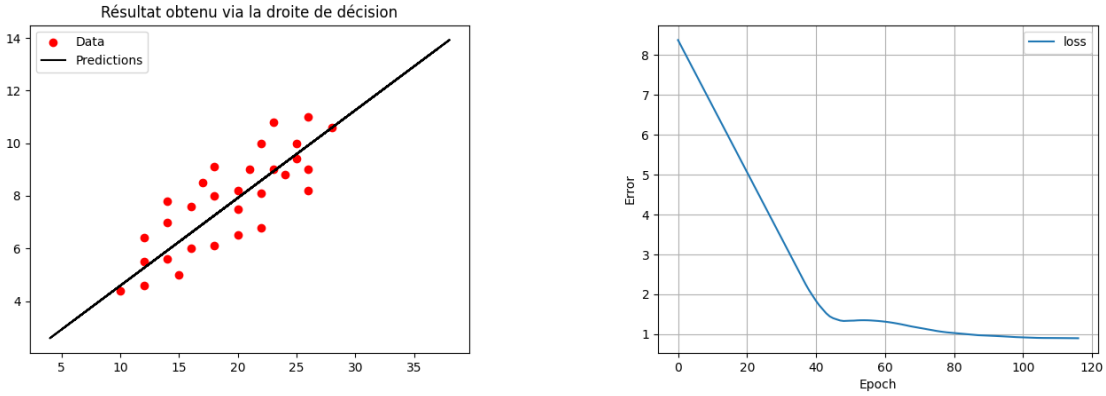

FIGURE 15 – modèle obtenu(lr=0.009,epochs=150) - valeurs fonction coût et itérations avant arrêt(loss=0.9,epochs=117)

1.4.1 Prédictions par rapport à des échantillons tirés aléatoirement sur l'intervalle [-10,100]

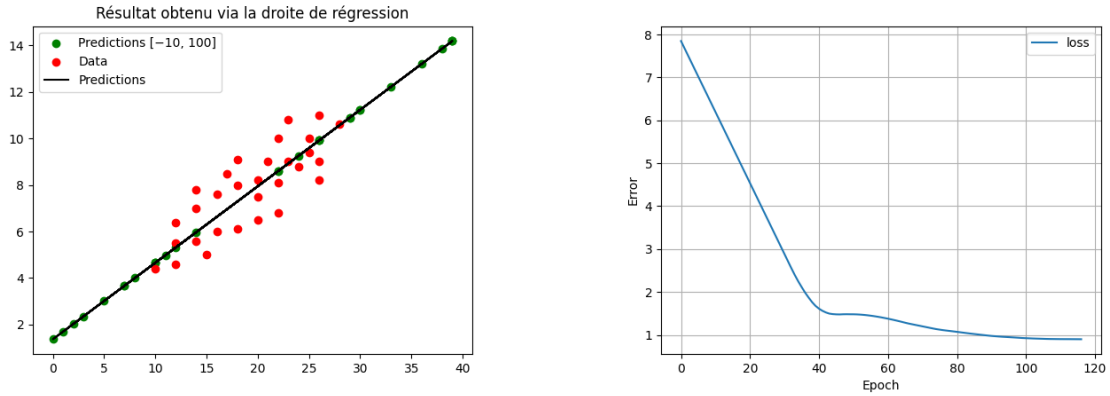

FIGURE 16 – modèle obtenu(lr=0.09,epochs=150,seuil=0.9) - valeurs fonction coût et itérations avant arrêt(epochs=117)

Paramètres utilisés durant l'apprentissage : il s'est avéré qu'un learning rate de `0.09` permettait de converger plus rapidement. De plus, le seuil de `0.9`, est le seuil considéré le plus efficace afin d'obtenir un modèle avec de bonnes prédictions et avec le moins d'itérations possibles : seulement 121. On remarque également sur la figure 16 que les prédictions par rapport aux échantillons tirés aléatoirement sur l'intervalle sont très satisfaisantes puisque les points sont correctement représentés sur la droite de régression.

2 Laboratoire 2

2.1 Classification de données

2.2 Exercice 1 - Classification de données avec de nombreuses entrées

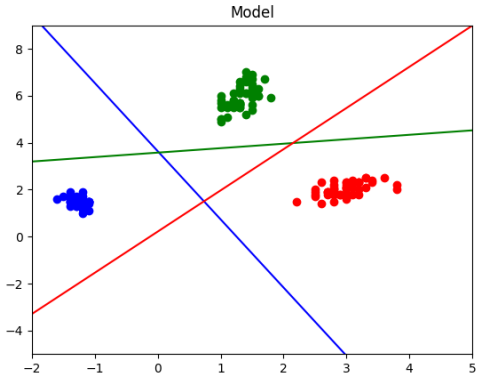

FIGURE 17 – Model obtenu et données d'apprentissage(n=3,lr=0.1, seuil=0.02, epochs_effectués=420)

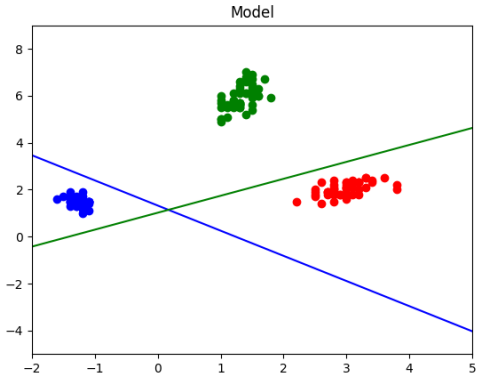

FIGURE 18 – Model obtenu et données d'apprentissage(n_entree=2,n_sortie=3,lr=0.1, seuil=0.02, epochs_effectués=289)

Afin d'obtenir un modèle correct, il a fallu au moins trois neurones de sortie. Effectivement, cela parait logique étant donné qu'il y a trois classes. Cependant, avec 2 neurones d'entrée, il est possible d'obtenir une classification correcte voir meilleur que celui à trois neurones (Figure 18) : effectivement, avec le modèle à trois neurones(Figure 17), il y a une zone d'incertitude où notre modèle ne pourra pas classifier les données.

Paramètres utilisés durant l'apprentissage : le learning rate retenu a été `0.1` car il permet de converger le plus rapidement (environ `60` itérations) vers le seuil choisi `0.002` qui est le seuil le plus bas qu'on puisse atteindre.

2.3 Exercice 2 - Classification de données avec de nombreuses entrées

2.3.1 Représentation des données

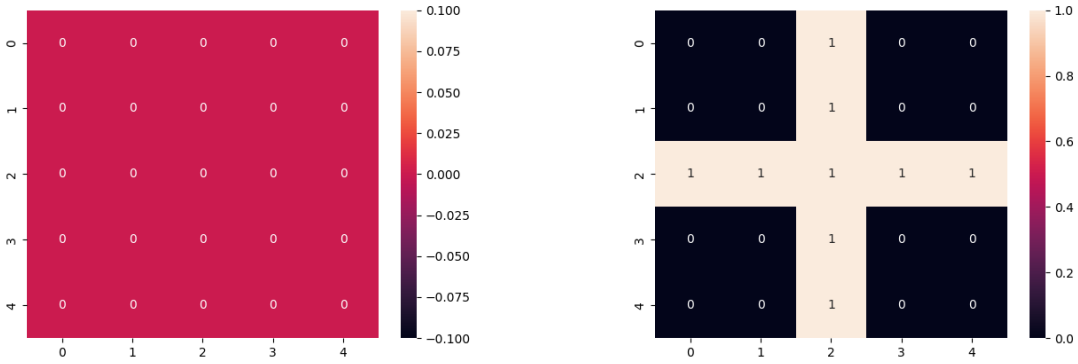

FIGURE 19 – Symbole 1 et 2

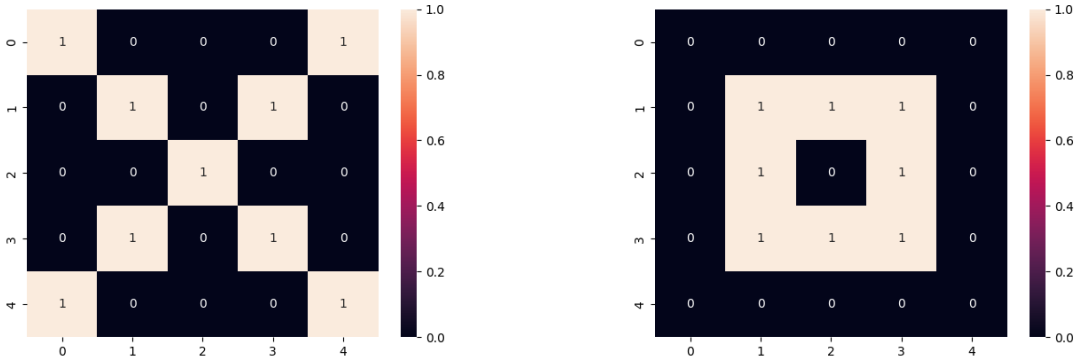

FIGURE 20 – Symbole 3 et 4

2.3.2 Représentation des poids donnés à chacun des pixels pour chacun des symboles.

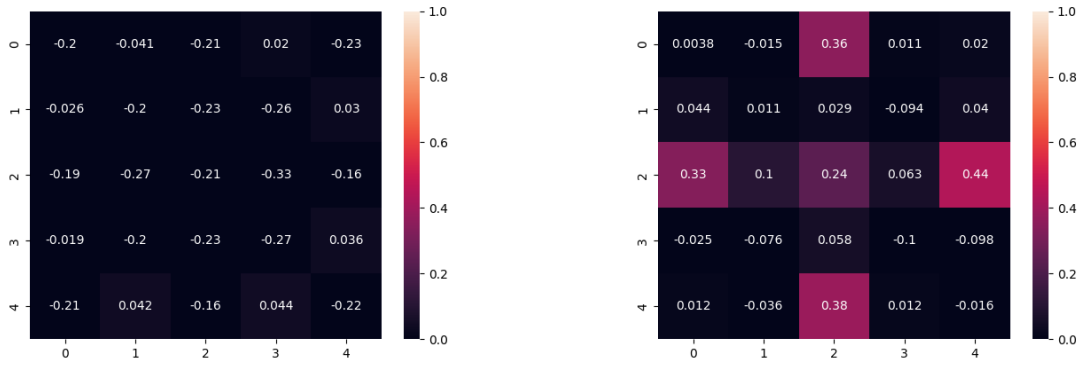

FIGURE 21 – Poids symboles 1 et 2

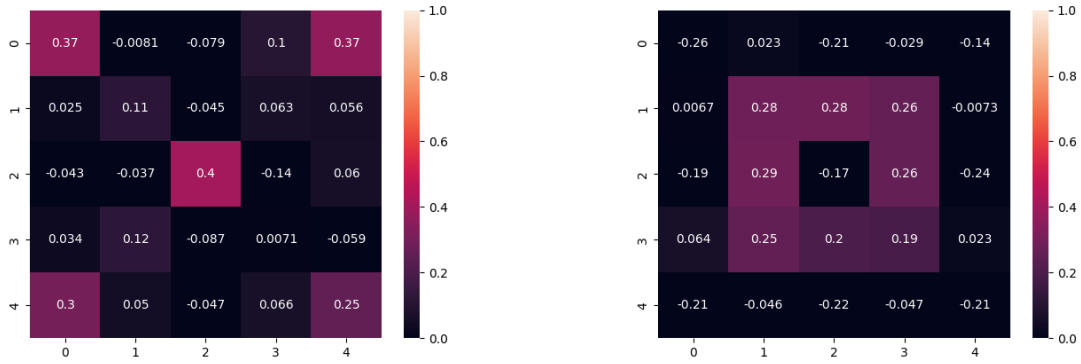

FIGURE 22 – Poids symboles 3 et 4

Justification : en comparant les graphiques obtenus avec les symboles (Figures 19 et 20), on s'aperçoit que l'on retrouve bien nos symboles : on remarque que les poids de valeurs assez supérieurs à 0 (à partir de 0.2), correspondent aux pixels de valeur 1 de nos symboles et les poids négatifs ou très proche de 0 correspondent à la valeur 0 de nos symboles.

2.3.3 Représentation du modèle et de l'évolution de la précision des prédictions

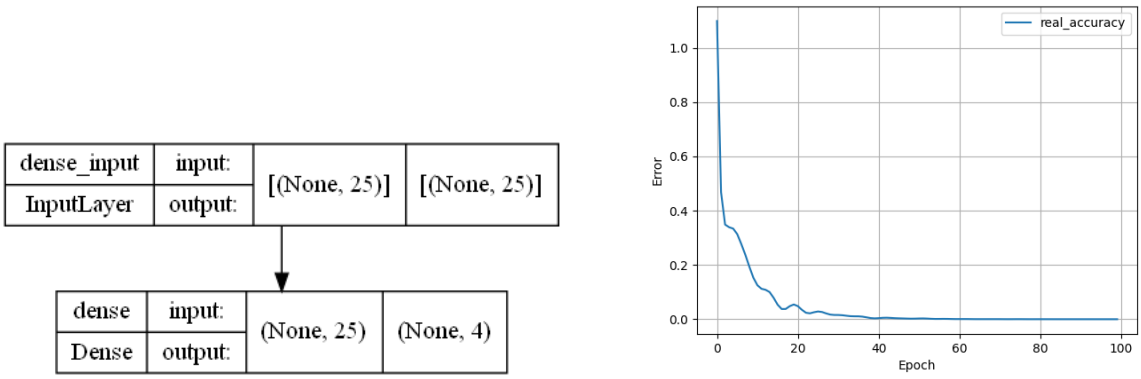

FIGURE 23 – Model et évolution de la métrique real_accuracy

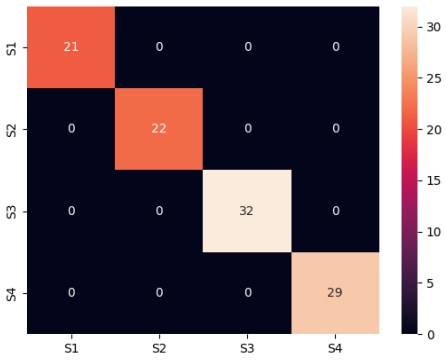

FIGURE 24 – Prédictions sur la table 3_5_noisy

Paramètres utilisés durant l'apprentissage : le learning rate choisi a été `0.1`, un nombre d'itérations de 100 et le critère d'arrêt a été la métrique real_accuracy : dès real_accuracy = 1.2e-05, on arrête l'apprentissage. **Qualité de la prédiction obtenue** : comme le montre la figure 24, notre modèle ne se trompe pas donc j'affirme que la qualité de la prédiction obtenue est excellente.

Méthode pour améliorer le résultat : a priori, à en juger par la qualité de la prédiction obtenue, le modèle n'a pas besoin d'être améliorer. Cependant, si je devrai proposer une méthode pour améliorer celui-ci, je dirai qu'on pourrait utiliser un modèle multi-couches.

2.4 Exercice 3 - Classification des Iris

2.4.1 Représentation des données à classifier

Les données ont été représentées à l'aide de deux graphiques. Le premier graphique est une représentation par pairplot :

```
1 sns.pairplot(data.iloc[:, 0:6], hue="species")
```

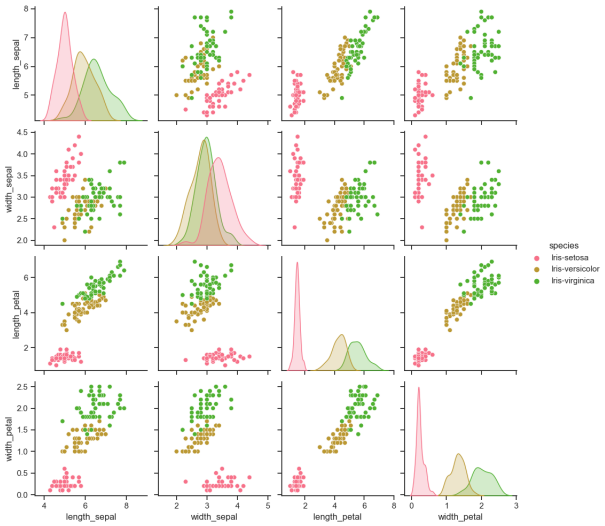

FIGURE 25 – Représentation des données par pairplot

Le second graphique est une représentation 3D réalisée à l'aide du package plotly-express :

```
px.scatter_3d(data, x="length_petal", y="width_petal", z="length_sepal", size="width_sepal",
    color="species")
```

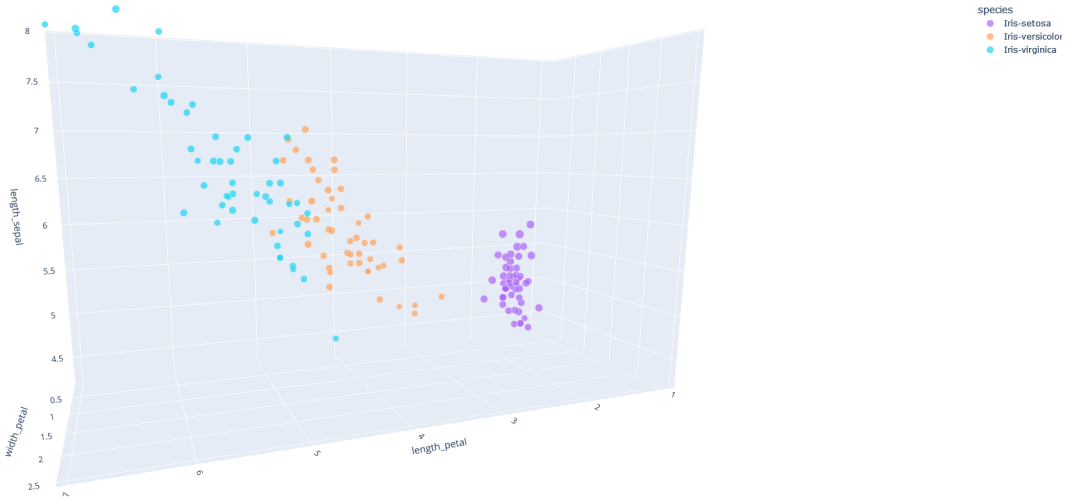

FIGURE 26 – Représentation des données en 3D

2.4.2 Entraînement avec 3 neurones

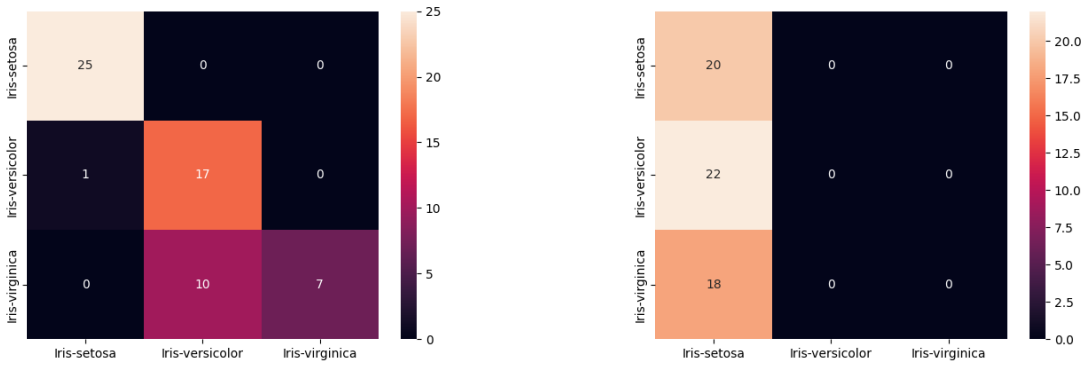

FIGURE 27 – Modèle à 1 neurone et modèle à 3 neurones

Justification : On s'aperçoit que le modèle avec trois neurones (Figure 27) commet beaucoup plus d'erreurs. Effectivement, la sortie attendue étant une seule valeur, le modèle à trois neurones de sorties n'arrive pas prédire correctement puisqu'il possède trois neurones de sorties.

2.5 Performances des différents modèles

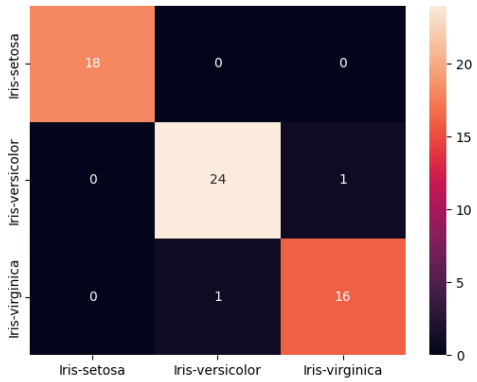

FIGURE 28 – Modèle monocouche de 3 neurones utile

Performances de mes différents modèles :

— Modèle un neurone : loss : 0.0554 - accuracy : 0.6333 - 1000 epochs

— Modèle trois neurones : loss : 0.0502 - accuracy : 0.3667

— Modèle monocouche de trois neurones utile : loss : 0.0359 - accuracy : 0.9889 - 25 epochs

Le modèle le plus efficace est donc le modèle monocouche de trois neurones utile puisqu'il n'a commis que 2 erreurs (Figure 28).

La principale différence avec le modèle monocouche de trois neurones utile (qui d'ailleurs est le plus performant avec 98% de précision), est que celui-ci utilise la fonction d'activation `softmax` (Figure 29). De plus, un callback personnalisé **EarlyStoppingAtGoodAccuracy(0.98)** a été utilisé afin d'arrêter l'apprentissage lorsque la précision dépasse les 98%.

*Note : la fonction d'activation **softmax** permet de transformer un vecteur réel en vecteur de probabilité. Cette fonction est généralement utilisé sur la couche de sortie d'un modèle pour faire de la classification multi-classes*

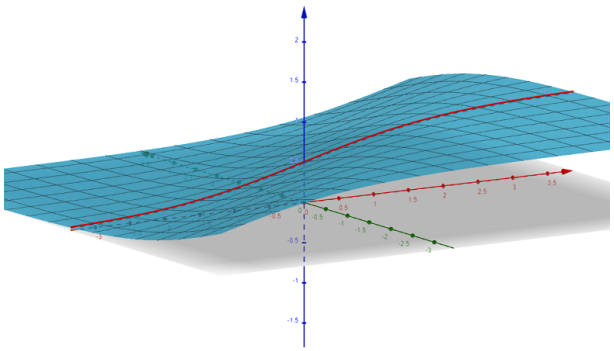

FIGURE 29 – Exemple de minimum local et minimum global

3 Laboratoire 3

3.1 Exercice 1 - Classification de données XOR

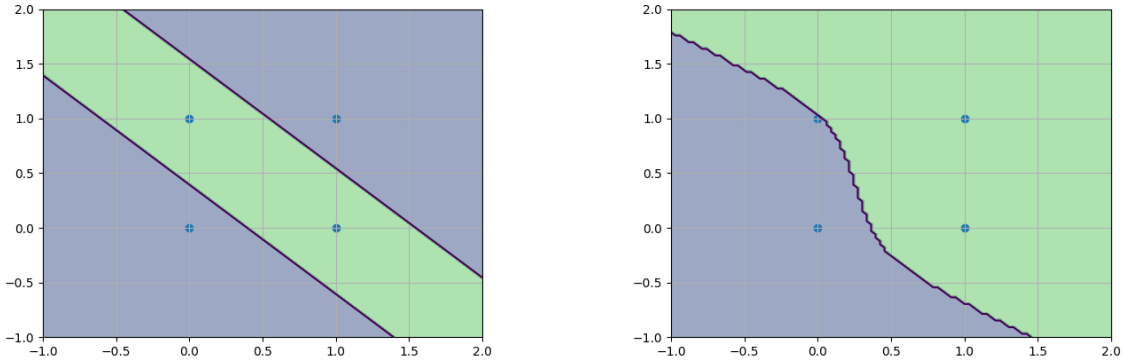

FIGURE 30 – Modèles obtenus

La métrique est-il utile dans notre cas ? Oui et non Dans notre cas, la métrique (ici la fonction coût) peut nous induire en erreur sur le nombre minimal d'erreurs : effectivement, il est possible que la valeur indiquée soit un minimum local (exemple Figure 31).

Effectivement, comme décrit dans la section 4.3 du livre théorique, s'il existe plusieurs minimums, alors notre réseau de neurones peut converger vers n'importe lequel de ces minimums notamment en fonction des poids qui sont initialisés aléatoirement tout en offrant un modèle acceptable et c'est ce qu'il se produit ici.

En conclusion, la métrique nous informe toujours sur le nombre d'erreurs cependant en fonction de l'initialisation des poids, on peut tomber sur différentes valeurs de la métrique tout en ayant un modèle capable de classifier nos données XOR.

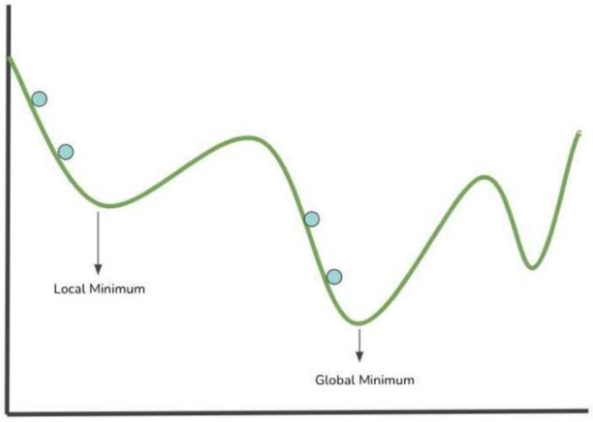

FIGURE 31 – Exemples minimums

3.2 Exercice 2 - Classification de données non-linéairement séparables

3.2.1 Modèles

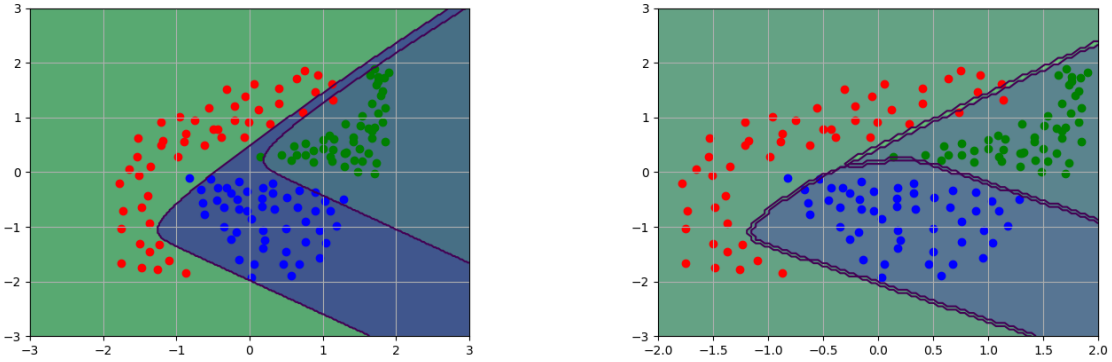

FIGURE 32 – Modèle 1(3-1-3) et Modèle 2(3-2-3)

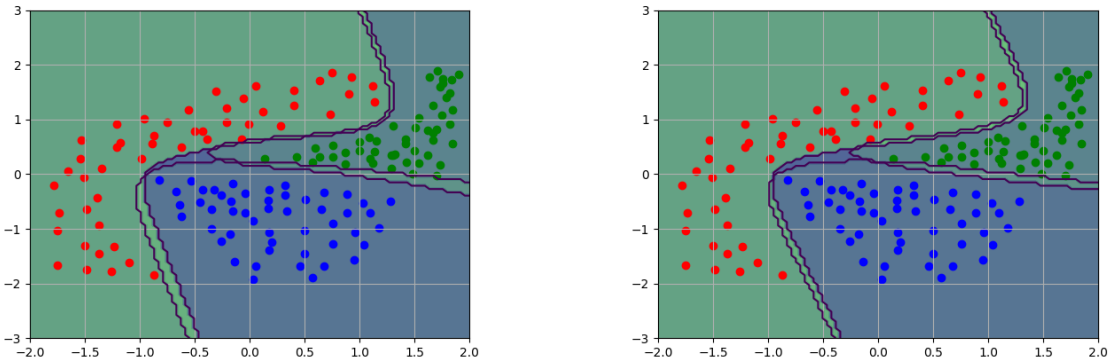

FIGURE 33 – Modèle 3(3-3-3) et Modèle 4(3-5-3)

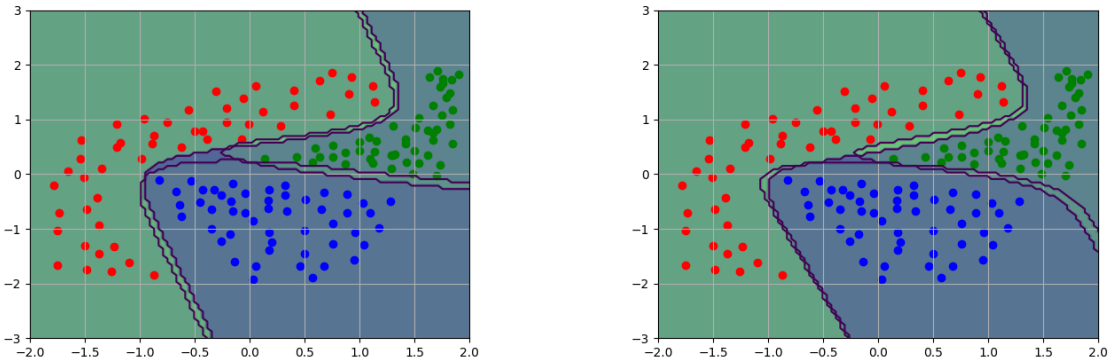

FIGURE 34 – Modèle 5(3-10-3) et Modèle 6(3-20-3)

Paramètres utilisés durant l'apprentissage : il s'agit des valeurs du cours théorique (à l'exception du taux d'apprentissage) :

— taux d'apprentissage : `0.1`

— nombre d'itérations maximum : `2000`

— seuil de tolérance : `0.001`

— stratégie d'apprentissage : Adam

3.2.2 Interprétation des zones de recouvrement et des zones vides

Ces zones sont des zones d'incertitude : le modèle n'est pas capable de décider parmi les 3 classes.

3.2.3 Variations du nombre de neurones

Table 1

C	nbIter	loss
1	2000	0.0790
2	2000	0.0288
3	484	9.7645e-04
5	250	9.8076e-04
10	249	8.9987e-04
20	221	8.3811e-04

Combien de neurones sont nécessaires pour une prédiction correcte ? Il faut au moins 3 neurones cachés afin d'obtenir une classification correcte. Cependant, 5 semble être la meilleur solution puisqu'il permet d'obtenir une classification correcte le plus rapidement possible. Pour finir, au delà de 5 neurones cachés, on voit que le modèle ne change pratiquement pas.

3.3 Exercice 3 - Classification des Iris

3.3.1 Variations de 2 à 5 couches

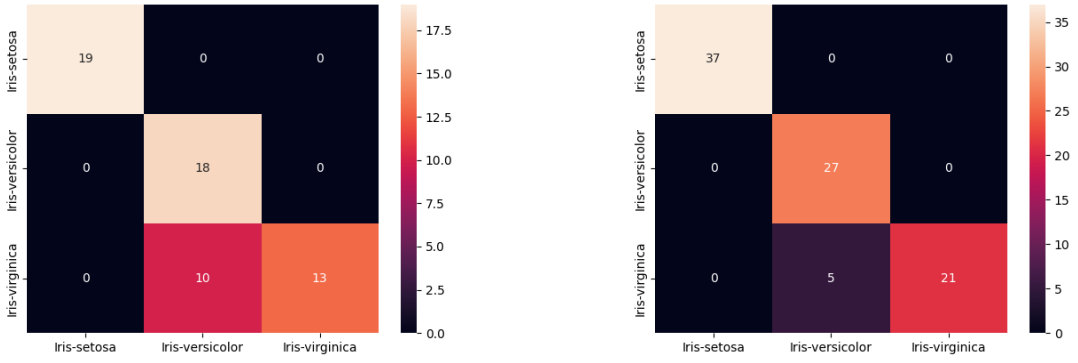

Figure 35 – N :3-3, loss : 0.1047 - accuracy : 0.8222 et N :3-10-3, loss : 0.0464 - accuracy : 0.9222

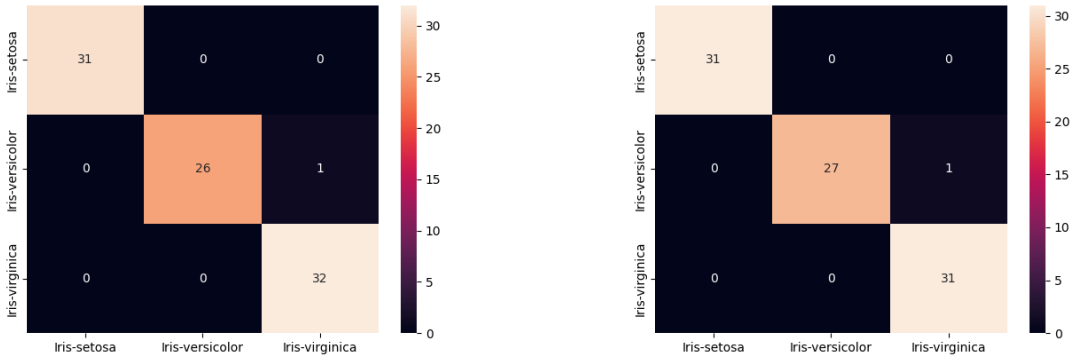

FIGURE 36 – N :3-10-10-3, loss : 0.0333 - accuracy : 0.9444 et N :3-10-10-10-3, loss : 0.0116 - accuracy : 0.9778

On remarque qu'à partir de 4 couches, le modèle ne commet plus qu'une erreur, là où avec moins de couches il commet trop d'erreurs.

Adam : il s'agit d'une amélioration de SGD. Contrairement à SGD qui utilise un taux d'apprentissage unique pour mettre à jour la valeurs des poids, Adam utilise un taux d'apprentissage qui est maintenu pour chaque poids et adapté tout au long de l'apprentissage en fonction de l'exemple traité. De cette manière, Adam converge généralement plus vite que SGD.

SparseCategoricalCrossentropy : il s'agit d'une fonction coût qui permet de calculer la perte d'entropie croisée entre les labels et les prédictions. Elle produit un index de la catégorie la plus probable. Il est généralement utilisé pour faire de la classification non binaire.

ReLU - Rectified Linear Unit : il s'agit d'une fonction d'activation qui permet de donner le maximum entre une valeur et 0. Autrement dit, l'utilisation de cette fonction est très appréciée puisqu'elle permet d'obtenir seulement des valeurs positives. C'est pourquoi elle est souvent utilisé dans les couches cachées mais jamais dans la couche de sortie.

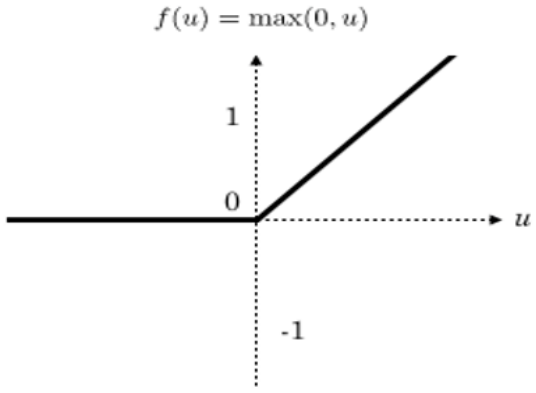

FIGURE 37 – Fonction d'activation relu