Open Access

Dynamic graph neural network

Réseau de neurones dynamique pour le clustering

Antonin LEFEVRE[1]

[1]Université de Reims

Abstract

Les réseaux de neurones dynamiques sont une branche peu développée du Deep Learning, qui repose sur un principe simple, l'architecture du réseau est dynamique. Cela a plusieurs avantages, notamment le fait que le réseau est en perpetuel apprentissage, et qu'il peut changer sa structure et le routage des informations en fonction des données, ce qui le rend très flexible. Le dynamisme s'opère au niveau de la profondeur du réseau mais aussi sur sa largeur (dans le cas d'un réseau en couches comme dans le modèle DAN2 [1]). Ici, on va encore un peu plus loin dans ce concept de réseau de neurones dynamiques, car on va utiliser une structure de graphe, c'est à dire sans organisation en couches comme les modèles classiques. Concernant ce projet, il a pour objectif de tester le pouvoir classificateur d'un réseau de neurones dynamiques en graphe décrit par l'article "A Dynamic Neural Network for Continual" [2]. L'article n'évoque qu'une partie mathématique et quelques voix pour la mise en place du modèle. Ainsi, La première étape de ce projet sera d'implémenter la structure du graphe ainsi que les méthodes associées telles qu'elles sont décrites dans cet article, puis, en fonction des resultats, d'améliorer le modèle. On verra que l'utilisation de la transformée de Fourier [3] et la transformée en ondelettes [4] sera nécessaire. Ensuite, une partie graphique sera implémentée avec la librairie Plotly qui servira à s'assurer de la bonne mise en place des premières méthodes (uniquement sur la première partie de l'implémentation). Concernant les phases d'expérimentation, on tentera pour commencer de classer plusieurs types de fonctions classiques, puis nous poursuivrons sur une classification de signaux sinusoïdaux, tout cela sans pré traitement des données, et nous tenterons de classer des chants d'oiseaux avec une autre méthode de calcul de distance. Ensuite, nous testerons la modification de nos données d'entrées avec une transformée de Fourier, ou encore une transformées en Ondelettes sur des électrocardiogrammes et des signaux sinusoïdaux. Dans ce repository, une première partie sera consacrée à l'aspect mathématique du modèle, pour mieux comprendre son fonctionnement. Puis sera expliqué l'implémentation avec Python avec les différents tests et résultats.

Key Points
- Théorie
- Implémentation
- Expérimentation

Correspondence to
Antonin LEFEVRE
antoninlefevre45@icloud.com

Data Availability
Tous les codes sont disponibles sur GitHub.

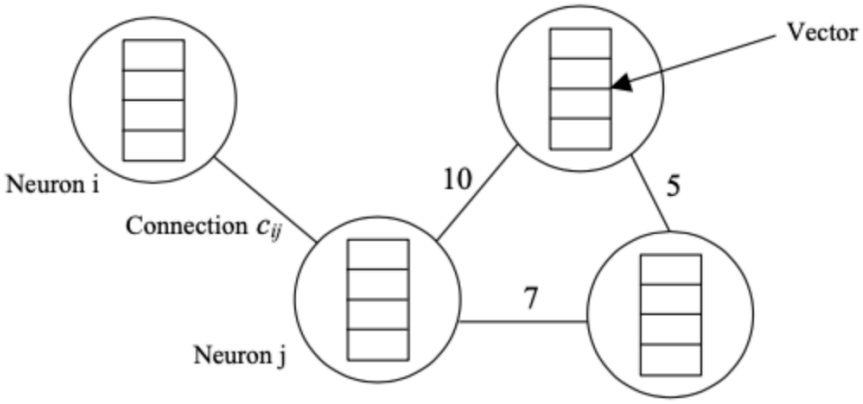
Keywords Réseaux dynamiques, Réseaux en graphe, Clustering

Table des matières

1.. Modèle mathématique

1.1.. Principe

Voici un réseau basique, chaque neurone x contient un vecteur qui est de la même taille que l'input. Les liaisons entre les neurones sont des scalaires, et on note $c_{\{ij\}}$ la liaison entre le neurone i et j. Les neurones les plus semblables sont connectés par un poids synaptique.



A partir de là, le vecteur d'entrée noté u est comparé avec le vecteur de chaque neurone. Le neurone le plus proche (avec une distance euclidienne notée d) de l'input est alors appelé le foyer, et est noté $z(x)$.

Ainsi, soit $x = [x_1, x_2, ..., x_n]^T$ un vecteur d'un neurone, et $u = [u_1, u_2, ..., u_n]^T$ le vecteur d'entrée, alors la distance euclidienne d entre x et u est définie par :

$$\|d\|_2 = \left[\sum_{i=1}^{m} (x_i - u_i)^2\right]^{\frac{1}{2}} \tag{1}$$

Donc

$$z(x) = \arg\min_j \|d\|_2 \tag{2}$$

Avec $j = 1, 2, 3, ..., l$ et l le nombre de neurones dans le graphe.

Le neurone d'entrée est alors connecté aux neurones dont la similarité dépasse un certain seuil.

La distance euclidienne entre le vecteur d'entrée u et le foyer z est ensuite utilisé pour modifier le foyer (son vecteur). On introduit le scalaire b_v un paramètre d'échelle qui correspond au learning rate du réseau.

$$\Delta z(x) = b_{v(z-u)} \tag{3}$$

Après modification du foyer, on va modifier de la même manière les neurones connectés à proximité du foyer (en dessous d'un certain seuil a_n de similarité), mais à un degré moindre par rapport au foyer. On introduit le scalaire b_c un paramètre d'échelle qui correspond au taux de changement du noeud. (k est le foyer)

$$\Delta x_j = b_c * c_{\{jk\}}(x_k - x_j) \tag{4}$$

Avec $k = 1, 2, 3, .., l$, $j \neq k$, $b_c \in R$.

On réduit aussi les connexions du foyers, ce qui rapproche les neurones similaires. La force avec laquelle elles sont actualisées est le scalaire b_l . La nouvelle valeur de la connexion entre j et k est alors $c_{\{jk\}} = b_{l(\|x_j - x_k\|)}$ Avec $b_l \in R$

Si l'entrée du réseau u est complètement différente des autres neurones (en terme de distance euclidienne) alors un nouveau neurone ou groupe de neurones est ajouté et connecté au foyer. Un neurone est ajouté quand $\|d\| > a_n$ avec $a_n \in R$ C'est à dire si la distance minimale entre l'entrée et les neurones dépasse le seuil a_n.

Élagage du réseau : On supprime les liens qui deviennent trop longs, c'est à dire soit a_r le seuil, le lien entre le neurone i et j est supprimé si $c_{\{ij\}} > a_r$.

Quand un neurone n'a plus de lien, il est supprimé.

1.2.. Prédiction

Il y a deux méthodes pour effectuer une prédiction

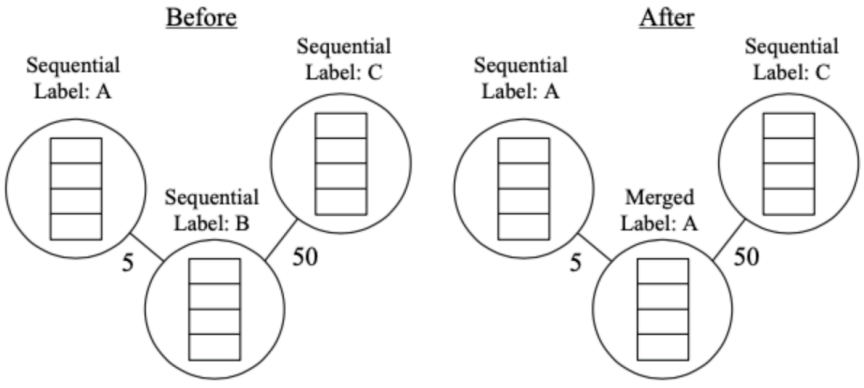
1.2.1.. ***Par concaténation**:*

En considérant que chaque neurone contient un vecteur qui contient en plus la sortie souhaitée. Ainsi, en envoyant en entrée un vecteur avec seulement une taille correspondant à la taille de ceux des neurones sans la sortie, le réseau devrait renvoyer la sortie du foyer associé.

Cette méthode induit que les données sont labellisés

1.2.2.. ***Par labellisation automatique des clusters**:*

Si la connexion entre deux neurones est suffisamment petite, le réseau va associer aux deux neurones le même label. Par exemple, sur l'image ci-dessous le neurone du Label A et le neurone du label B ont une liaison de poids 5, ce qui veut dire qu'ils sont très similaires, dans ce cas les labels sont fusionnés.



Marche très bien pour des données non labélisées. On utilisera cette dernière méthode de prédiction dans ce projet.

2.. Implémentation

L'implémentation repose sur la création de deux classes. Une classe représentant les neurones (`Neuron`), et une classe représentant le graphe (`Graph`). Ainsi, chaque instance de graphe possède un certain nombre de neurones.

La classe `Neuron` possède plusieurs paramètres :

- vecteur : qui représente le vecteur du neurone, c'est sur ce vecteur que repose le modèle
- config : qui contient les seuils du modèle ainsi que la taille des signaux si nécessaire
- index : identifiant unique d'un neurone dans un graphe, il est attribué grâce à un compteur interne au graphe
- label : c'est la classe à laquelle le neurone appartient, il est attribué lors de l'ajout des neurones
- liaisons : qui est un dictionnaire des liaisons dont les clés représentent l'index d'un neurone, et la valeur son poids

La classe `Graph` possède également plusieurs paramètres :

- neurons : qui est un dictionnaire contenant tous les neurones du graphe, indexé par l'index des neurones
- compt_neurons : qui est initialisé à 0 lors de la création du graphe et qui correspond au compteur de neurones, pour l'attribution des index
- fct_distance : technique de calcul de la distance entre les neurones, par défaut la distance euclidienne

Ces paramètres seront fixes tout au long de ce projet. Concernant les méthodes de ces deux classes, elles seront détaillées par la suite.

La première étape de la modélisation est la création du graphe et l'ajout de neurones. On définit alors la méthode `addNeuron` de la classe `Graph` prenant en paramètre un objet de la classe `Neuron`.

On définit dans cette méthode 3 cas :

- Si le graphe est vide : le neurone prend comme label son index, et aucune liaison n'est alors créée.
- Si le graphe contient un seul neurone : on assigne au nouveau neurone le label du premier si la distance entre les deux est inférieure au seuil `an`, sinon son label est défini par son index. On crée ensuite la liaison entre les deux. (qui est ajouté aux deux neurones)
- Si il y a plus que deux neurones, on calcul le foyer du nouveau neurone. Si la distance entre les deux est inférieure au seuil `an` il prend le label du foyer, et on connecte au nouveau neurone tous les autres à une distance inférieure à `an` . Sinon, l'index du nouveau neurone devient aussi son label, et il n'est connecté qu'a son foyer.

Dans le modèle initial proposé par l'article, après chaque ajout d'un neurone on doit, si le neurone tout juste ajouté est à une distance inférieure à $a_{\{n\}}$ de son foyer, modifier le foyer ainsi que toutes ces liaisons et neurones voisins. Si une liaison devient supérieure à $a_{\{r\}}$ durant cette modification alors la liaison est supprimée. (tous les voisins du foyer sont déjà par définition à une distance inférieure à $a_{\{n\}}$)

On définit alors trois méthodes dans la classe `Neuron` qui vont permettre ces modifications :

- `alterFoyer` : qui va altérer le vecteur du foyer du nouveau neurone ajouté
- `alterVoisins` : qui va modifier les voisins du foyer du nouveau neurone ajouté, selon le modèle mathématique
- `alterLiaisons` : qui va altérer les liaisons du foyer du nouveau neurone ajouté selon le modèle mathématique, et supprimer celles qui deviennent supérieures à `ar`

Si un neurone n'a plus de connexion on lui attribut son label comme classe. (il constituera à lui seul un cluster)

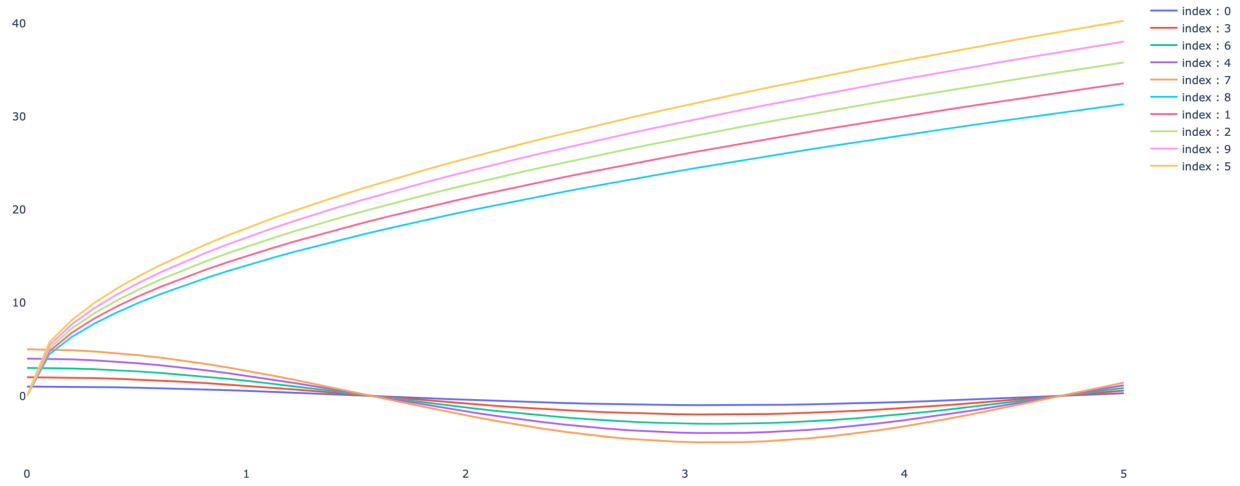
Dans l'implémentation on ajoute une méthode `fit()` qui prend en paramètre un dictionnaire indexé sur les entiers naturels, contenant le tableau de valeurs de chaque neurone. Elle ajoute automatiquement les neurones avec la méthode `add_Neuron`.

Un problème dans l'implémentation de la fonction d'affichage du graphe apparaît, en effet, l'ajout d'un neurone assez proche de son foyer (distance inférieure à $a_{\{n\}}$) induit une modification du foyer et des voisins et liaisons de ce dernier. Ceci déséquilibre le lien mathématique (de distance euclidienne) entre les neurones et de ce fait, la méthode permettant d'afficher le graphe ne permettra pas de le faire. On se basera ainsi sur l'affichage des neurones (avec la méthode `__repr__` de chaque classe) du graphe avec leur label pour savoir comment le modèle les a rassemblés.

3.. Expérimentations

3.1.. Classification de fonctions classiques

On va dans cette première partie utiliser le modèle de la façon la plus basique possible. La distance euclidienne utilisée par le réseau nous oblige à avoir des fonctions avec le même nombre de points. Les signaux que nous comparerons seront uniquement soumis à la distance euclidienne. Prenons un ensemble de 10 neurones, dont les index **0, 3, 4, 6, 7** sont ceux représentants des fonctions cosinus (en bas) et **1, 2, 5, 8, 9** des fonctions racines (en haut). On peut les représenter graphiquement :



On obtient après ajout de ces neurones le graphe suivant : (on affiche la liste des neurones du graphe, les vecteurs ne sont pas affichés pour des raisons de lisibilité)

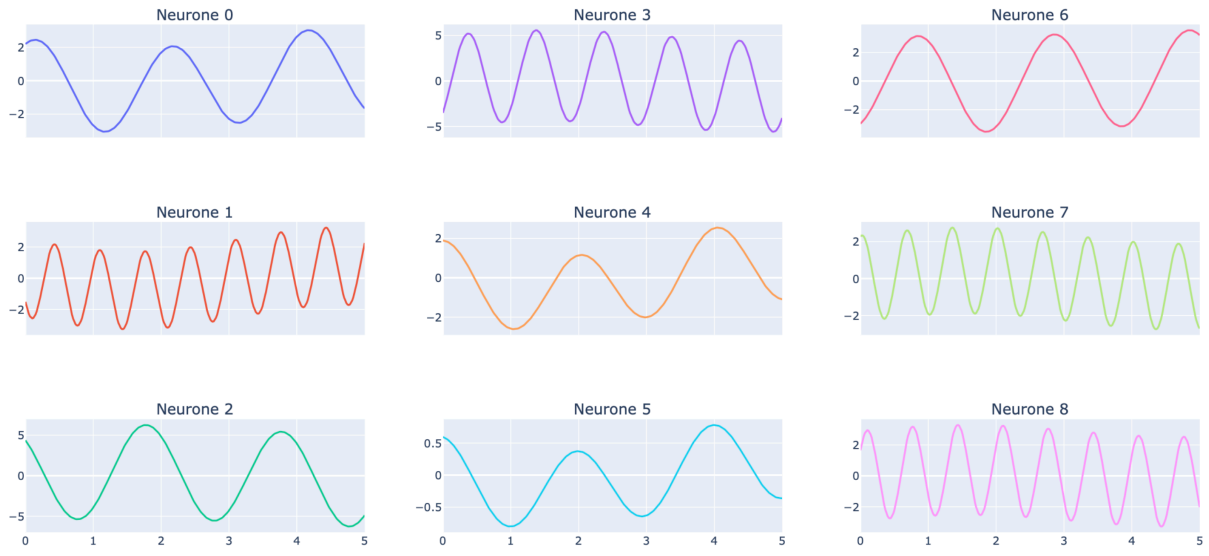
```
{
0: Neuron(index=0, liaisons={1: 15.04025}, label=0),
 1: Neuron(index=1, liaisons={0: 15.04025, 2: 0.9881875, 5: 6.126875, 8: 2.33225, 9: 18.5865},
label=1),
 2: Neuron(index=2, liaisons={1: 0.9881875, 3: 58.147}, label=1),
 3: Neuron(index=3, liaisons={2: 58.147, 4: 3.4405, 6: 2.06425, 7: 22.845}, label=3),
 4: Neuron(index=4, liaisons={3: 3.4405, 6: 6.976, 7: 24.364}, label=3),
 5: Neuron(index=5, liaisons={1: 6.126875, 6: 17.707, 7: 47.834}, label=1),
 6: Neuron(index=6, liaisons={3: 2.06425, 4: 6.976, 5: 17.707, 7: 10.631}, label=3),
 7: Neuron(index=7, liaisons={3: 22.845, 4: 24.364, 5: 47.834, 6: 10.631}, label=3),
 8: Neuron(index=8, liaisons={1: 2.33225, 9: 37.861}, label=1),
 9: Neuron(index=9, liaisons={1: 18.5865, 8: 37.861}, label=1)
 }
```

On remarque que les neurones d'index 1, 2, 5, 8 et 9 sont ajoutés à la même classe. Ce groupe de 5 neurones correspond exactement au type de fonction racine, donc le réseau a parfaitement réussi à regrouper ces données ensemble. De même que les neurones 3, 4, 6, 7 qui sont les neurones de type cosinus. Mais dans cette dernière classe le neurone 0 a été classé en tant qu'outlier. (ce n'est pas totalement absurde dans le sens ou il sa fonction est éloignée de toutes les autres)

On va poursuivre les tests avec d'autres données pour voir comment le modèle se comporte.

3.2.. Classification de signaux sinusoïdaux

Dans cette deuxième partie, on va continuer à utiliser uniquement la distance euclidienne tel que décrit dans l'article. On prend alors 9 neurones, qui représentent des signaux quelconques qui sont des sommes aléatoires de fonctions sinusoïdales. On va alors tester différents seuils pour voir si on arrive à trouver une classification satisfaisante. On peut déjà tracer les courbes représentant les 9 neurones :



On remarque des signaux de différentes périodicités, avec des amplitudes plus ou moins grandes. On va maintenant ajouter nos neurones au réseau pour voir comment le modèle va les rassembler. Voici le résultat :

```
{
0: Neuron(index=0, vecteur="", liaisons={1: 21.28756, 2: 33.84723999999999, 4: 6.818349999999999,
5: 23.8938}, label=0),
 1: Neuron(index=1, vecteur="", liaisons={0: 21.28756, 3: 67.593}, label=1),
 2: Neuron(index=2, vecteur="", liaisons={0: 33.84723999999999}, label=2),
 3: Neuron(index=3, vecteur="", liaisons={1: 67.593, 6: 77.476}, label=3),
 4: Neuron(index=4, vecteur="", liaisons={0: 6.818349999999999}, label=0),
 5: Neuron(index=5, vecteur="", liaisons={0: 23.8938}, label=0),
 6: Neuron(index=6, vecteur="", liaisons={3: 77.476, 7: 35.44029999999999}, label=6),
 7: Neuron(index=7, vecteur="", liaisons={6: 35.44029999999999, 8: 17.2781}, label=7),
 8: Neuron(index=8, vecteur="", liaisons={7: 17.2781}, label=7)
 }
```

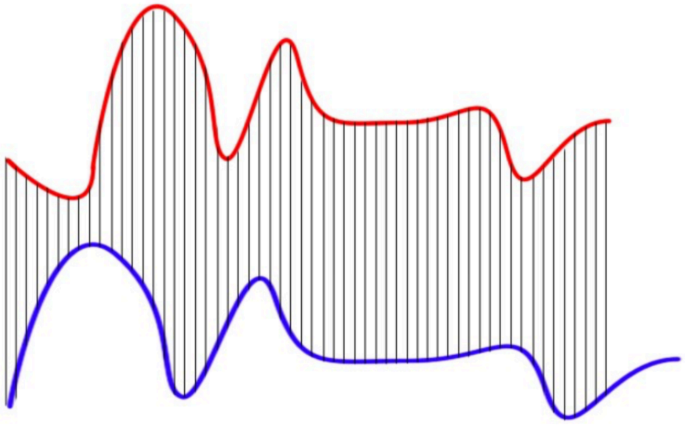
On remarque que certains neurones on été ajoutés au même ensemble. Les neurones 0, 4 et 5 appartiennent au même cluster, de même que les neurones 7 et 8. Enfin, les autres neurones sont classés dans des clusters différents. On peut relever que le modèle a rassemblé les signaux qui se superposent bien, cependant il ne prend pas en compte le fait que les signaux sont périodiques et que deux signaux peuvent se superposer à une translation près. On peut alors comprendre pourquoi le réseau a réussi à classer de façon correcte les fonctions racines et sinus, c'est parce qu'elles sont superposables.

On pourrait alors essayer de représenter ces signaux d'une autre manière, qui pourrait faire ressortir les caractéristiques de ces derniers pour mieux les comparer .. avec **une décomposition en séries de Fourier**.

3.3.. Classification de chants d'oiseaux avec la méthode Dynamic Time Warping

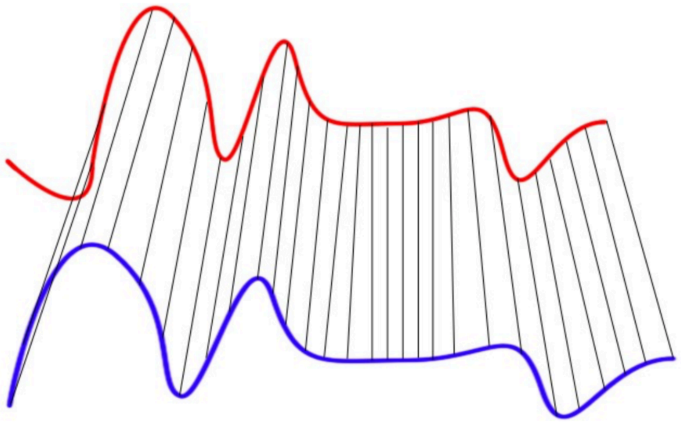
Dans cette section, nous allons changer la façon de calculer les distances à l'interieur du réseau, mais nous n'utiliserons pas de preprocessing sur les donnés, les signaux restent bruts. En effet, la distance euclidienne ne convient que pour des signaux de même taille. Si on calcule la distance euclidienne entre un vecteur de taille `n` et un autre de taille `m` tel que `n<m` alors cela revient à calculer la distance entre deux vecteur de taille `n`. (le vecteur de taille `m` est tronqué)

On peut représenter graphiquement la distance euclidienne comme ceci :



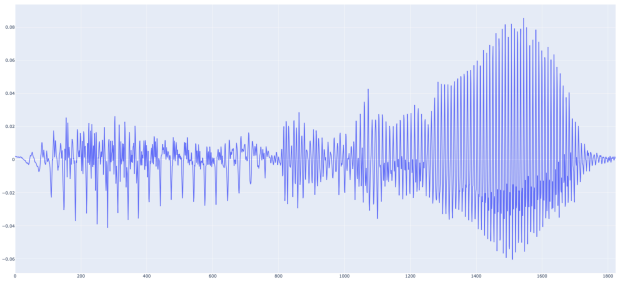
Le signal rouge est le vecteur de taille `n`, il est plus petit que le signal bleu de taille `m`. On peut alors remarqué que la distance euclidienne entre les deux sera grande, car les deux signaux, malgrès leur ressemblance, ne sont pas alignés. Et la distance euclidienne compare les valeurs une à une dans l'ordre. Ils sont alignés à une translation près, comme ce qu'on avait remarqué dans la partie 2.

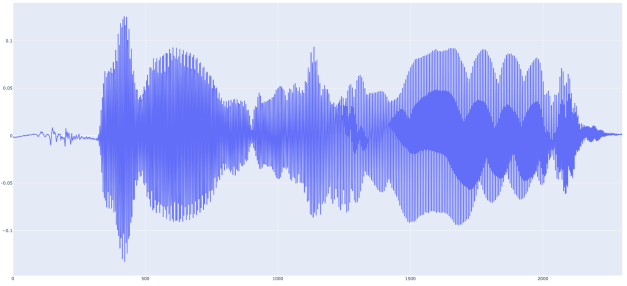
Pour palier à ce problème, il existe la méthode **Dynamic Time Warping**. Cette méthode permet de trouver l'alignement global optimal entre deux signaux, c'est-à-dire d'associer chaque élément de chaque signal à au moins un élément de l'autre signal en minimisant les coûts d'association. Le coût d'une association correspond à la distance entre les deux éléments. Le résultat numérique fournit par DTW correspond à la somme des hauteurs des "barreaux" formés par les associations (les barres noires entre les signaux rouge et bleu). On remarque sur la figure ci-dessous à gauche des signaux que DTW a réaligné correctement les deux signaux, et parvient ainsi à saisir des similarités que la distance euclidienne ne peut extraire.



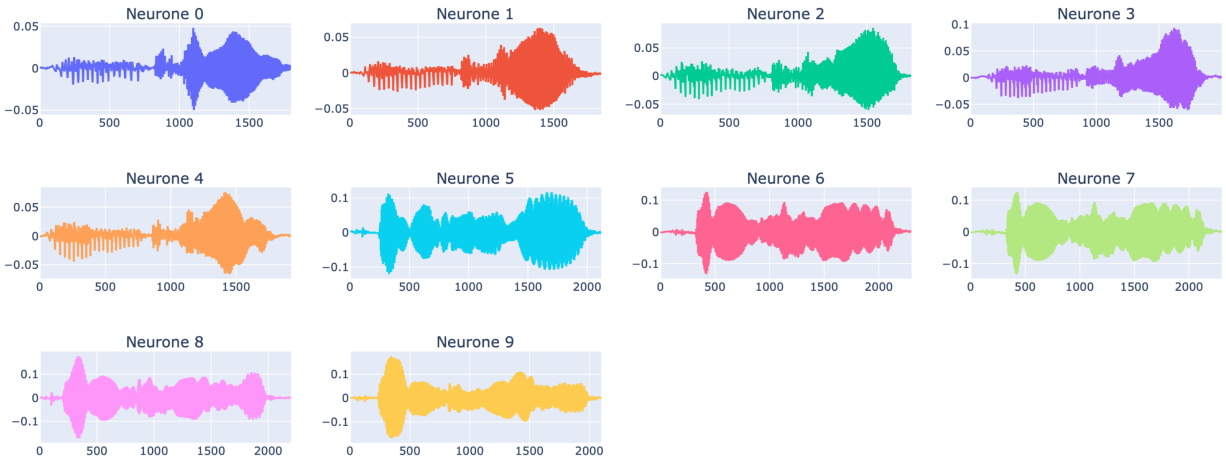
Cette fois-ci nous allons utiliser de vraies données, qui sont des chants d'oiseaux. On isolera les syllabes de plusieurs espèces, ce qui constituera nos données d'entrées. Puis, on passera au modèle ces données qui utilisera la méthode DTW pour calculer les distances.

Voici des exemples de syllabes de chants d'oiseaux:

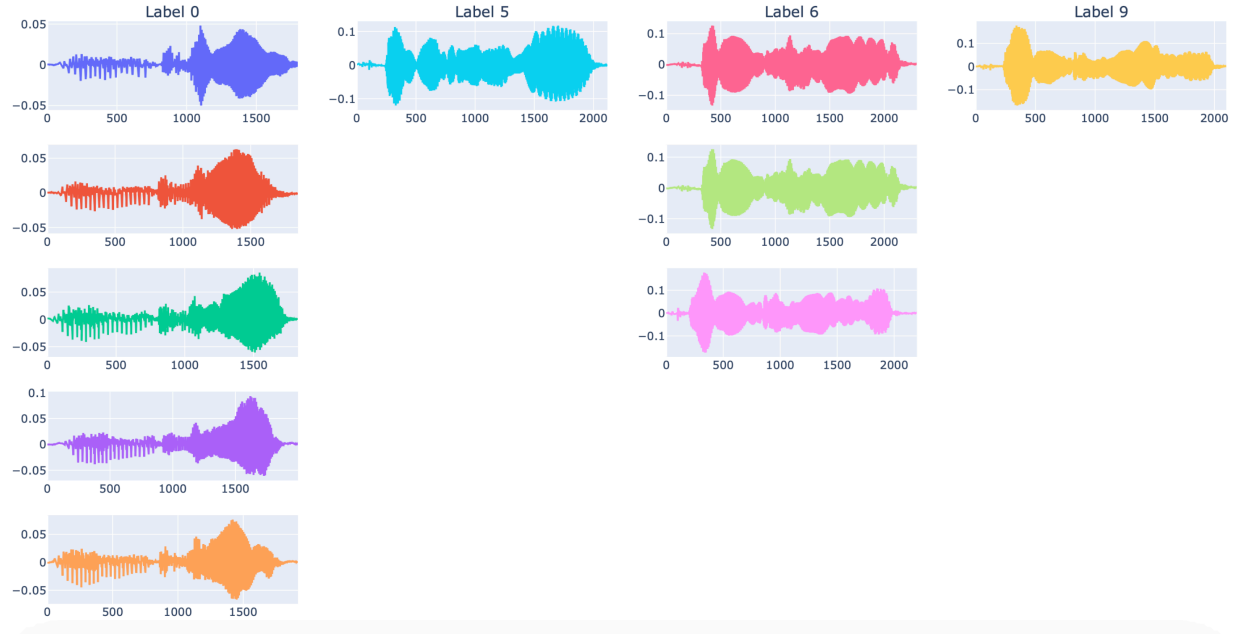




On remarque que les deux syllabes sont de tailles différentes, c'est ce qui motive l'utilisation du DTW. Maintenant, voici le dataset de syllabes de chants d'oiseaux que nous allons utiliser dans la suite des tests:



Les neurones 0 à 4 contiennent des syllabes d'une espèce d'oiseau, et les neurones 5 à 9 contiennent des syllabes d'une autre espèce. On ajoute maintenant les neurones au réseau, et on observe la classification du modèle :

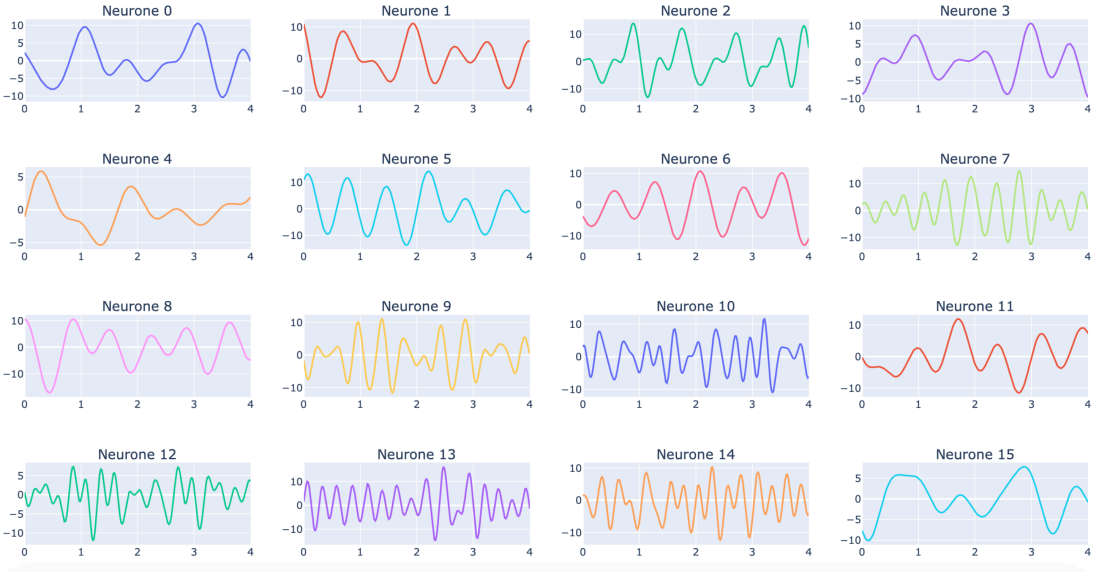


Tout d'abord, on s'aperçoit que toute la première espèce d'oiseau a été associée au même cluster (de label 0). Pour la deuxième espèce, le résultat est plus mitigé, en effet l'espèce a été divisée en trois sous-catégories (labels 5, 6 et 9). En fait, notre modèle a un pouvoir de classification trop élevé pour nos données. Il cherche a vraiment trouver les différences entre les signaux. Mais, la classification par le modèle reste stable, en effet après avoir essayé une multitude de seuils différents le réseau ne classe quasiment jamais deux signaux d'espèces différentes ensemble, ce qui est très encourageant en terme de véracité du modèle.

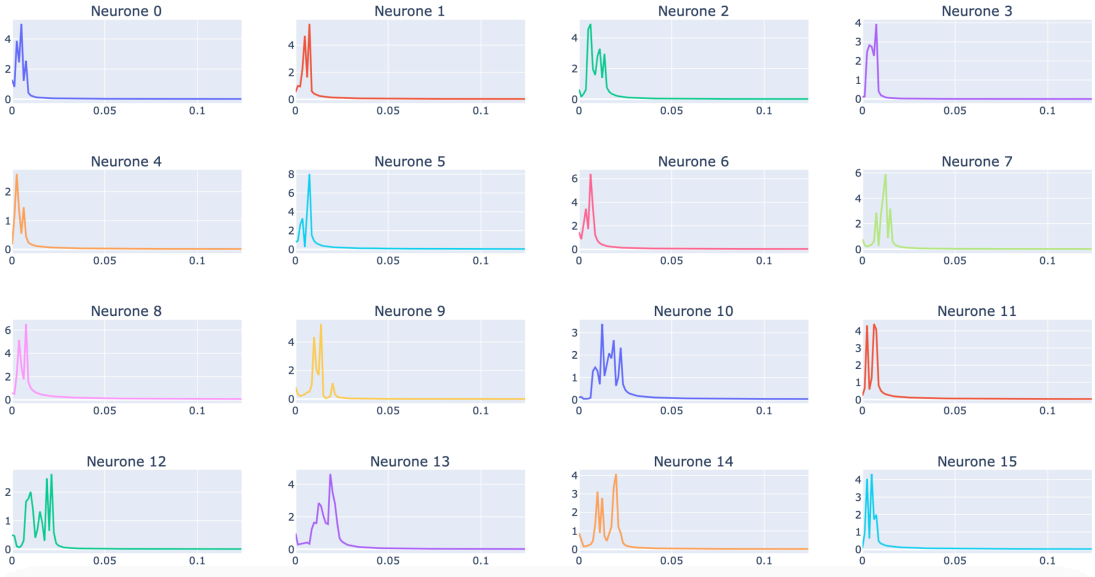
3.4.. Classification de signaux sinusoïdaux soumis à une transformée de Fourier

On va dans cette section utiliser la transformée de Fourier pour voir si le modèle réussi à mieux classer les signaux. On va prendre comme précédemment des signaux sinusoïdaux aléatoires. Le principe est donc le suivant : on va effectuer une transformée de Fourier sur chacun des signaux brutes, et le résultat de chacune des transformations est alors passé aux neurones. Cette manipulation va permettre au réseau de ne pas être trompé entre deux signaux en moyenne identiques, et parfaitement superposables. C'est donc entre les transformées de Fourier des signaux que le modèle va faire les calculs de distance euclidienne.

Voici les 16 signaux, correspondant aux 16 neurones du réseau :



On observe à vue d'oeil des différences au niveau des fréquences. Appliquons maintenant une transformée de Fourier sur chacun d'entre eux. Voici les résultats :



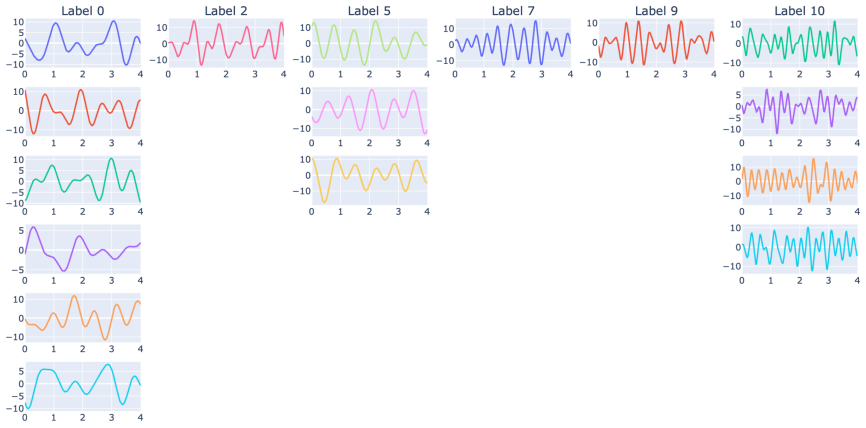
Ce sont ces signaux qui seront passés aux neurones. On rappelle que soit f notre signal, alors sa transformée de Fourier et la fonction $F(f)$ définie par :

$$F(f) = \int_{-\infty}^{+\infty} f(x) e^{-\mathrm{ix}\,\xi}\,\mathrm{dx} \tag{5}$$

Si on regarde les données brutes du réseau on obtient ceci :

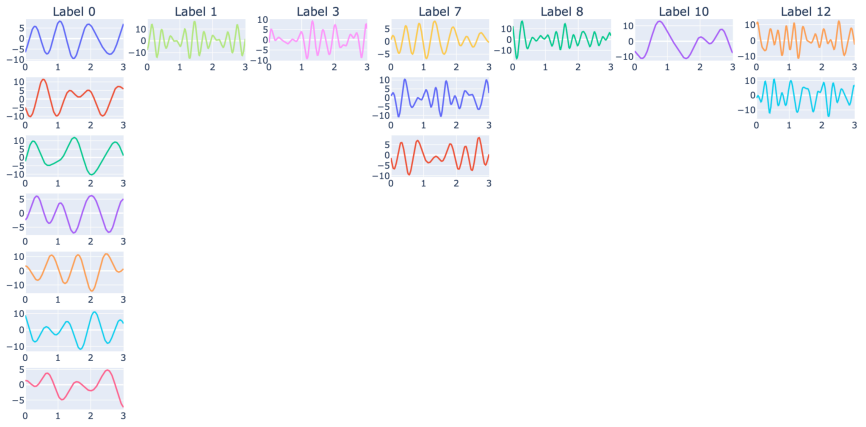

```
{
0: Neuron(index=0, liaisons={1: 0.006926400000000001, 3: 0.164, 4: 1.0906, 7: 2.1456, 8:
1.1800000000000002, 11: 1.1814, 15: 3.769}, label=0),
 1: Neuron(index=1, liaisons={0: 0.006926400000000001, 2: 0.30952, 3: 0.028696000000000006, 4:
1.1492000000000002, 8: 1.2998, 11: 1.2300000000000002, 15: 0.5498000000000001}, label=0),
 2: Neuron(index=2, liaisons={1: 0.30952, 15: 6.163}, label=2),
 3: Neuron(index=3, liaisons={0: 0.164, 1: 0.028696000000000006, 4: 1.0162000000000002, 5:
0.32352000000000003}, label=0),
 4: Neuron(index=4, liaisons={0: 1.0906, 1: 1.1492000000000002, 3: 1.0162000000000002, 8: 5.77,
11: 5.981, 15: 3.842}, label=0),
 5: Neuron(index=5, liaisons={3: 0.32352000000000003, 6: 0.21260000000000004, 8: 0.9372, 11: 6.406}
, label=5),
 6: Neuron(index=6, liaisons={5: 0.21260000000000004, 8: 4.875, 11: 6.383}, label=5),
 7: Neuron(index=7, liaisons={0: 2.1456, 9: 7.783, 10: 0.26996000000000003}, label=7),
 8: Neuron(index=8, liaisons={0: 1.1800000000000002, 1: 1.2998, 4: 5.77, 5: 0.9372, 6: 4.875, 11:
6.268, 15: 4.79}, label=5),
 9: Neuron(index=9, liaisons={7: 7.783}, label=9),
 10: Neuron(index=10, liaisons={7: 0.26996000000000003, 12: 0.03649600000000001, 13:
1.0204000000000002, 14: 1.0712}, label=10),
 11: Neuron(index=11, liaisons={0: 1.1814, 1: 1.2300000000000002, 4: 5.981, 5: 6.406, 6: 6.383, 8:
6.268, 15: 4.869}, label=0),
 12: Neuron(index=12, liaisons={10: 0.03649600000000001, 13: 0.979, 14: 5.743}, label=10),
 13: Neuron(index=13, liaisons={10: 1.0204000000000002, 12: 0.979, 14: 5.648}, label=10),
 14: Neuron(index=14, liaisons={10: 1.0712, 12: 5.743, 13: 5.648}, label=10),
 15: Neuron(index=15, liaisons={0: 3.769, 1: 0.5498000000000001, 2: 6.163, 4: 3.842, 8: 4.79, 11:
4.869}, label=0)
 }
```

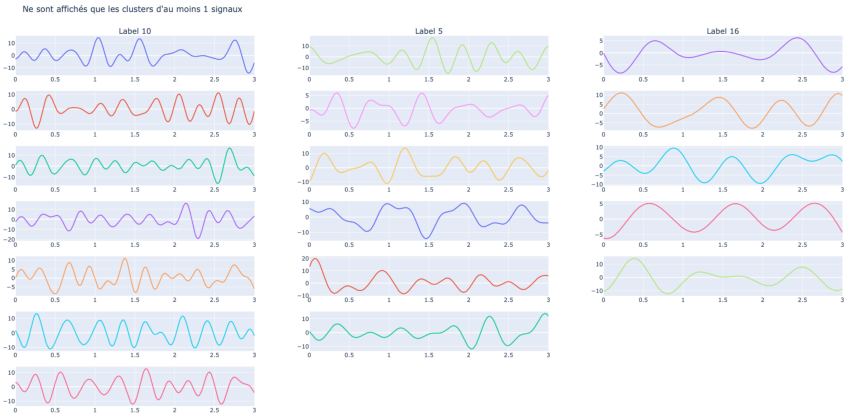
De façon plus lisible, voici comment le réseau a classé les signaux par label (1 colonne = 1 cluster de neurones) :

Clusters	Signaux
Label 0 : 0 1 3 4 11 15 Label 2 : 2 Label 5 : 5 6 8 Label 7 : 7 Label 9 : 9 Label 10 : 10 12 13 14	

On peut ainsi remarquer que cette fois ci, la classification est plutôt très bien réussie. Les neurones de label 0 sont les signaux avec la fréquence la plus basse, et les neurones de label 10 sont les signaux de plus hautes fréquences. Les labels intermédiaires sont des signaux de fréquences moyennes (par rapport aux labels 0 et 10), et donc la classification est un peu plus compliquée.

On peut réitérer la classification avec d'autres données, voici certains résultats :

Clusters	Signaux
Label 0 : 0 2 4 5 6 13 14 Label 1 : 1 Label 3 : 3 Label 7 : 7 9 11 Label 8 : 8 Label 10 : 10 Label 12 : 12 15	

Clusters	Signaux
Label 10 : 10 17 9 0 3 6 12 Label 5 : 5 15 14 11 1 8 Label 16 : 16 13 7 2 4	

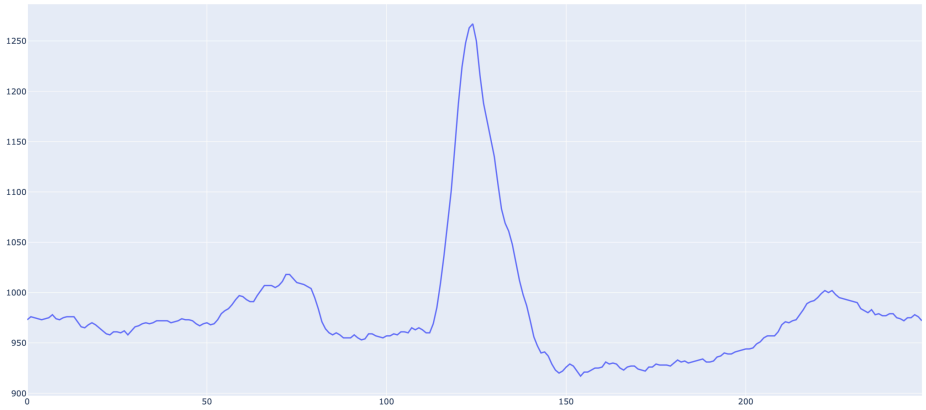
Il n'y a pas d'erreurs aberrantes, contrairement aux configurations précédentes, et ce sur une multitude de données. On peut alors valider cette technique (d'application du FFT) comme étant efficace, et aussi valider le modèle décrit dans l'article sur lequel se base ce projet. (tout en ajoutant cette petite subtilité dans le passage des signaux aux neurones)

Cependant, ce modèle a une limite dans son implémentation actuelle (telle que décrite dans l'article). En effet, tout est basé sur la distance euclidienne entre les neurones, ce qui immplique que les signaux doivent impérativement être de la même taille.

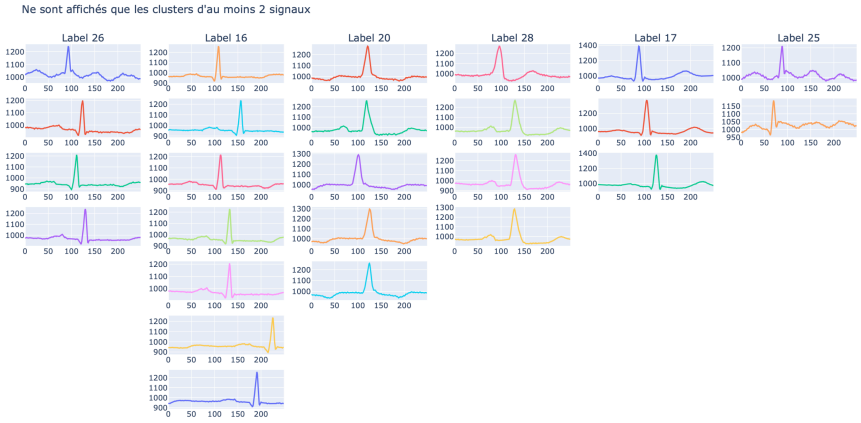
3.5.. Classification d'électrocardiogrammes soumis à une transformée de Fourier

À partir de cette section nous utiliserons des électrocardiogrammes pour tester notre réseau. Nous avons extrait 30 cycles cardiaques qui appartiennent à 3 classes de dysfonctionnement cardiaque différentes. Ces cycles sont indéxés de 0 à 29. Les cycles de 0 à 9 correspondent à des cycles normaux, les cycles 10 à 19 correspondent à un battement auriculaire prématuré et les cycles 20 à 29 à un flutter atrial.

Voici à quoi ressemble un cycle normal :



En faisant passer nos données dans le réseau voici le résultat. Chaque colonne représente un cluster, et chaque courbe est un cycle cardiaque.

Clusters	**Signaux**
===== Résultat de la classification Label 26 : 26 6 15 8 Label 16 : 16 11 10 7 9 13 12 Label 29 : 29 Label 20 : 20 0 22 21 24 Label 28 : 28 3 2 4 Label 17 : 17 19 18 Label 25 : 25 27 Nombre de neurones supprimés : 4	

Les deux premiers clusters ne sont pas vraiment optimum, mais tous les suivants le sont à une erreur près. Le réseau est relativement efficace pour associéer des ECG de même nature. À noter que pour une pathologie cardiaque, tous les cycles ne sont pas forcément différents des cycles normaux, ce qui peut expliquer les erreurs. Par exemple, le neurone 20 (le premier du label 20) ressemble beaucoup au neurone 6 (le deuxième du label 26) alors que le neurone 20 correspond à la pathologie du flutter atrial et le neurone 6 à un cycle normal.

3.6.. Classification d'électrocardiogrammes soumis à une transformée en Ondelettes

Dans cette section, nos cycles cardiaques des électrocardiogrammes seront modifiées à l'aide de la transformée en Ondelettes. La transformation par les ondelettes est une transformation des fonctions/signaux plus performante que celle de Fourier car elle est notamment capable de détecter les portions du signal qui varient plus rapidement que d'autres.

La décomposition d'une fonction en ondelettes consiste à l'écrire comme une somme pondérée de fonctions obtenues à partir d'opérations simples effectuées sur une fonction principale appelée ondelette-mère. Ces opérations consistent en des translations et dilatations de la variable. Selon que ces translations et dilatations sont choisies de manière continue ou discrète, on parlera d'une transformée en ondelettes continue ou discrète. Ici, nous n'utiliserons que la transformation en ondelettes unidimensionnelle et continue.

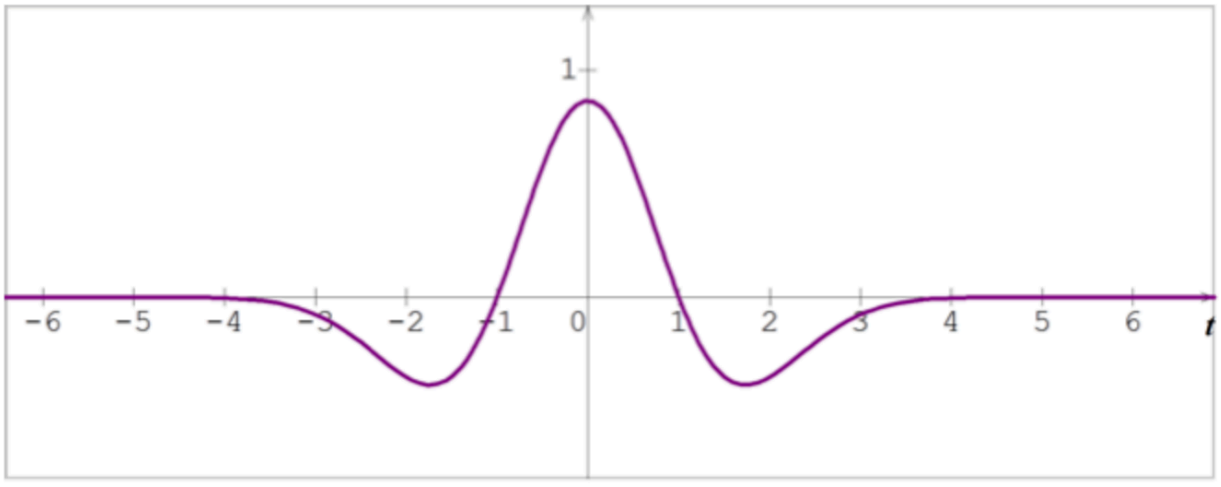
La transformée en ondelettes continue utilise des dilatations et des translations de la fonction ondelette mère ψ. La transformée en ondelettes continue de la fonction f est définie à facteur constant près comme le produit scalaire de f et de ψ.

L'ondelette mère que nous utiliserons ici est l'ondelette **"chapeau mexicain"** définit par :

$$\psi : \mathbb{R} \to \mathbb{R} \tag{6}$$

$$t \to \frac{2}{\sqrt{3}} \pi^{-\frac{1}{4}} (1 - t^2) e^{-\frac{t^2}{2}} \tag{7}$$

Et qui ressemble à ceci :



En faisant passer les données dans le réseau nous obtenons ces résultats. Chaque colonne représente un cluster, et chaque courbe est un cycle cardiaque.

Clusters	Signaux
`` ===== Résultat de la classification Label 26 : 26 23 29 Label 16 : 16 11 10 19 Label 1 : 1 7 20 9 0 21 3 Label 5 : 5 2 24 18 4 Label 28 : 28 17 Label 13 : 13 6 12 14 15 8 Label 22 : 22 Label 25 : 25 27 ```	

Les performances sont meilleures qu'avec la transformée de Fourier, il y a moins d'erreur.

4.. RÉSULTATS

Nous avons donc réussi à développer dans un premier temps un modèle qui classifie des fonctions classiques (cosinus et racine). Nous avons ensuite utilisé la transformée de Fourier pour permettre au réseau de ne pas se tromper sur des signaux ressemblant à une translation près en entrainant un réseau sur des signaux sinusoïdaux puis sur des ECG, en observant des résultats plutot satisfaisant. Nous avons également utilisé la transformée en ondelette, qui est plus performante que la transformée de Fourier, pour entrainer un réseau sur les ECG qui est légerement plus performant que le modèle utilisant la transformée de Fourier. Nous avons enfin tenté d'utiliser la méthode DTW (sans prè-traitement de données) qui permet de calculer la distance entre 2 signaux de tailles différentes, mais à cause de son temps d'execution, on ne l'utilisera pas.

Durant les tests, on remarque que si les données sont rangées dans l'ordre lors du `fit()` alors les neurones se suivent dans la classification finale, et donc les résultats sont faussés. Ainsi, on veillera à mélanger les données avant la phase d'aprentissage. De plus, on ajoutera une normalisation des données pour accélerer les calculs. Ces méthodes sont ajoutées dans la class `Graph` qui permet de normaliser et mélanger les données.

Quelques problèmes apparaissent, en effet le modèle utilise un certains nombre de seuils, qui ne peuvent qu'être définie de façon empirique selon les données. Cela impose donc, lors de l'utilisation du modèle.

5.. Bonus

Cette partie est un bonus qui a été developpé dans un premier temps pour afficher un réseau de neurones en graphe lorsque les distances entre neurones ne tiennent compte **uniquement** des distances euclidiennes entre ces derniers. Dans le modèle détaillé dans ce projet, les neurones sont modifiés, et donc on ne peut pas les représenter avec cette méthode.

Les données que nous avons à disposition sont les neurones avec leur vecteur. Nous calculerons toutes les distances nécessaires. Le but est alors de générer les coordonnées des neurones pour pouvoir les afficher avec Plotly. On se ramène à un problème purement mathématique, **comment placer `n` points en ne connaissant que les distances entre eux**. Pour cela on va utiliser une méthode geométrique consistant à trouver l'intersection de n cercles, grâce à un système à n équations non linéaires. Voici les étapes de l'algoritme :

- Étape 1: On place le premier neurone à la position ($x = 0$, $y = 0$)
- Étape 2: Le deuxième neurone est translaté sur l'axe des x par rapport au premier neurone, ainsi en notant `d` la distance entre les deux neurones, le deuxième neurone est alors en position (d, 0)
- Étape 3: Pour les autres neurones, on résout un système de n équation à 2 inconnues qui permet de trouver le neurone à l'intersection des cercles dont tous les autres points en sont les centres.

Prenons l'exemple ou nous avons les 2 premiers neurones A et B, et nous voulons ajouter un 3e neurone C :

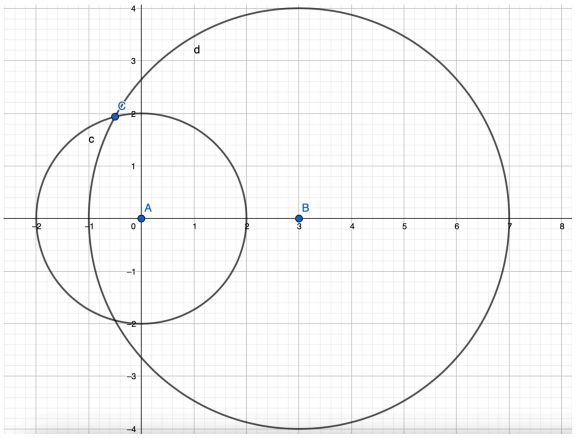
Le premier point A est en coordonnées ($x_1 = 0$, $y_1 = 0$) avec comme rayon $r_1 = 2$ qui est la distance entre A et le neurone C à ajouter Le deuxième point B est en coordonnées ($x_2 = 3$, $y_2 = 0$) avec comme rayon $r_2 = 4$ qui est la distance entre B et C

Ainsi, mathématiquement pour trouver les coordonnées (x, y) des intersections entre les 2 cercles de centre A et B voici le système :

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 - r_1^2 = 0 \\ (x - x_2)^2 + (y - y_2)^2 - r_2^2 = 0 \end{cases}$$

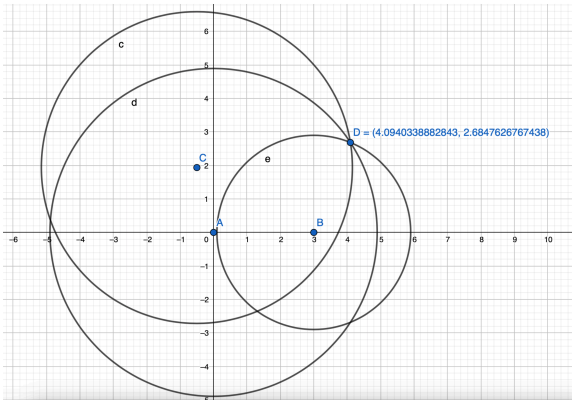
Pour choisir quel point prendre, on laissera la fonction de scipy nommée fsolve choisir.

On obtient mathématiquement :

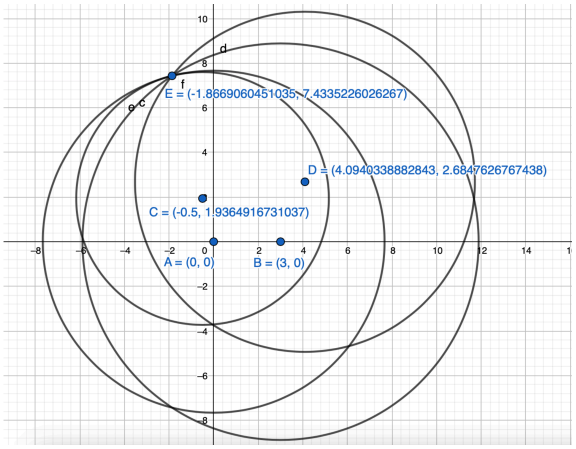
Mathématiquement	**Géométriquement**
``In [3]: x = [0, 3] ...: y = [0, 0] ...: r = [2, 4] ...: print(solve_inter_circles(x, y, r)) ...: (-0.4999999999999999, 1.9364916731037085)```	

Puis, pour chaque nouveau neurone à ajouter, on ajoute une équation au système, ce qui nous donne les coordonnées (x, y) du nouveau neurone. Voici les résultats sur les deux prochains points :

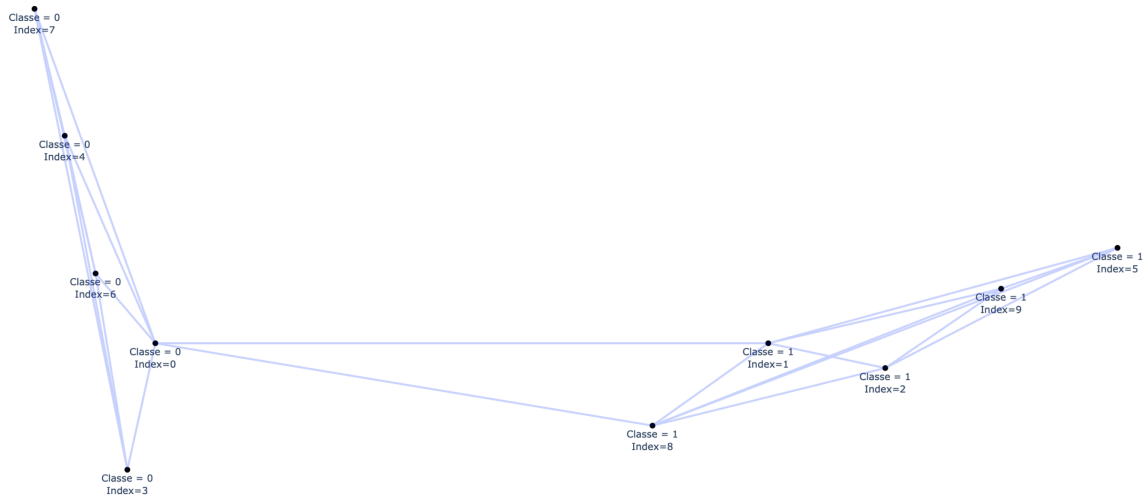
On cherche les coordonnées du point D :

Mathématiquement	**Géométriquement**
``In [8]: x = [0, 3, -0.4999999999999999] ...: y = [0, 0, 1.9364916731037085] ...: r = [4.8958210862793, 2.8991137920321, 4.6545737572407] ...: print(solve_inter_circles(x, y, r)) ...: (4.09403388828438, 2.6847626767436994)```	

On cherche les coordonnées du point E :

Mathématiquement	**Géométriquement**
In [9]: x = [0, 3, -0.4999999999999999, 4.09403388828438] ...: y = [0, 0, 1.9364916731037085, 2.6847626767436994] ...: r = [7.6643718897902, 8.8850454548993, 5.6644312314894, 7.6212548653834] ...: print(solve_inter_circles(x, y, r)) ...: (-1.8669060451034436, 7.433522602626717)	

Voici un exemple de résultat obtenu avec 10 neurones :



Les 5 neurones à gauche représentent des fonctions racines, et les 5 neurones de droites représentent des fonctions sinus. Cette représentation est seulement basée sur la distance euclidienne entre les neurones. Comme ces fonctions sont très distinctes, les neurones sont bien séparés. Mais comme vu dans ce projet, la distance des fonctions n'est pas suffisante pour avoir un modèle de classification performant.

6.. Conclusion

Ce projet nous a ammené, au travers de cette branche des réseaux de neurones dynamique, à découvrir de nouveaux concepts et de nouvelles représentations des données. On peut dire que ce type de réseau en graphe est très efficace pour classer des signaux notamment, et aussi pour détecter des outliers (des anomalies). Mais, comme les résultats l'ont montré, il ne faut pas seulement travailler avec les signaux brutes, mais plutôt avec leur transformée de Fourier, ou encore mieux, les transformées en ondelettes, qui permet au réseau de généraliser les types de signaux de façon plus efficace et sans faire d'erreur grossière.

References

[1] J. Velásquez, C. Franco, and Y. Morales, “A Review of DAN2 (Dynamic Architecture for Artificial Neural Networks) Model in Time Series Forecasting”, *Ingeniería y Universidad*, pp. 135–146, 2012.

[2] R. Lang, K. Warwick, and R. England, “A Dynamic Neural Network for Continual”, 2002.

[3] Wikipedia, “Transformation de Fourier”. [Online]. Available: https://fr.wikipedia.org/wiki/Transformation_de_Fourier

[4] Wikipedia, “Décomposition en ondelettes”. [Online]. Available: https://fr.wikipedia.org/wiki/Ondelette