

PFE

Khaled MEDJKOUH
Davidson Lova RAZAFINDRAKOTO

13 avril 2023

Table des matières

1	Introduction	3
1.1	Réseau de neurones <i>feedforward</i>	3
1.2	Source d'incertitudes dans un réseaux de neurones	4
1.2.1	Acquisition des données	4
1.2.2	Structure du modèle	4
1.3	Prédiction	5
1.3.1	Incertitude aléatoire	5
1.3.2	Incertitude épistémique	5
2	Réseau de neurones bayésiens (BNN)	6
2.1	Méthodes variationnelles	6
2.2	Méthode de Laplace	6
2.3	Méthodes par échantillonnage ou Monte Carlo	7
3	Algorithme BNN-ABC-SS	8
3.1	ABC (<i>Approximate Bayesian Computation</i>)	8
3.2	SS (<i>Subset Simulation</i>)	8
3.3	Pseudo - Code	9
4	Réalisation	11
4.1	Cosinus perturbé	11
4.2	Sinus perturbé	12
4.3	Maybe more	15
5	Conclusion et perspective	16

1 Introduction

Aujourd'hui, les applications des modèles/algorithmes d'apprentissage machine à base de réseaux de neurones arrivent à accomplir des tâches variées et de plus en plus complexes. Ce sont des modèles paramétriques souvent avec un nombre très importants de paramètres à ajuster pendant la période d'entraînement. En fonction des conditions de l'entraînement du modèles (données d'entraînement, choix d'hypeparamètres), ces paramètres ne seront pas les mêmes et naturellement la sortie du modèle sera affecté par cette variabilité. Comme la fonction du modèle est de sortir la bonne sortie quand on introduit une entrée, il y a un besoin de savoir contrôler cette variabilité pour avoir une confiance dans le résultat du modèle. Ce travail va tenter de reproduire un des méthodes qui tentent à contrôler cette variabilité, proposée dans cet article [5], l'algorithme BNN-ABC-SS.

1.1 Réseau de neurones *feedforward*

Un réseau de neurones *feedforward* est une fonction non linéaire paramétrée. La fonction prend au niveau de sa couche d'entrée un vecteur d'entrée $x \in \mathcal{X}$, le transforme à travers les couches cachées pour enfin sortir un vecteur de sortie $y \in \mathcal{Y}$ à la couche de sortie.

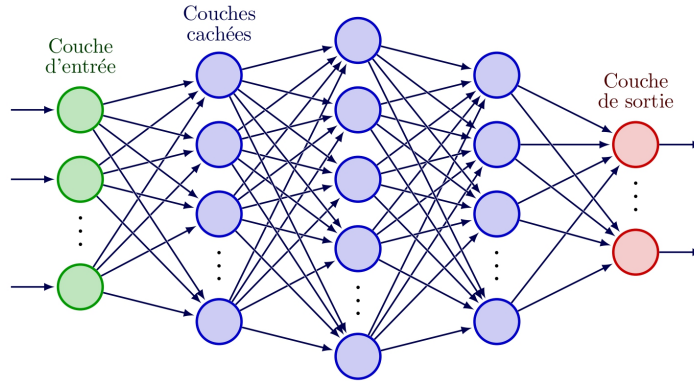


FIGURE 1 – Schéma de réseaux *feedforward* [1]

Pour passer d'une couche à une autre chaque neurone $a_s^{(j+1)}$ de la couche d'arrivée reçoit une contribution des neurones $\{a_i^{(j)}\}_{i=1}^n$ de la couche de départ pondérés par les poids $\{w_{i,s}^{(j)}\}_{i=1}^n$. Cette contribution sera biaisé (avec $b_s^{(j)}$) ensuite transformée par une fonction qu'on appelle *fonction d'activation* noté σ .

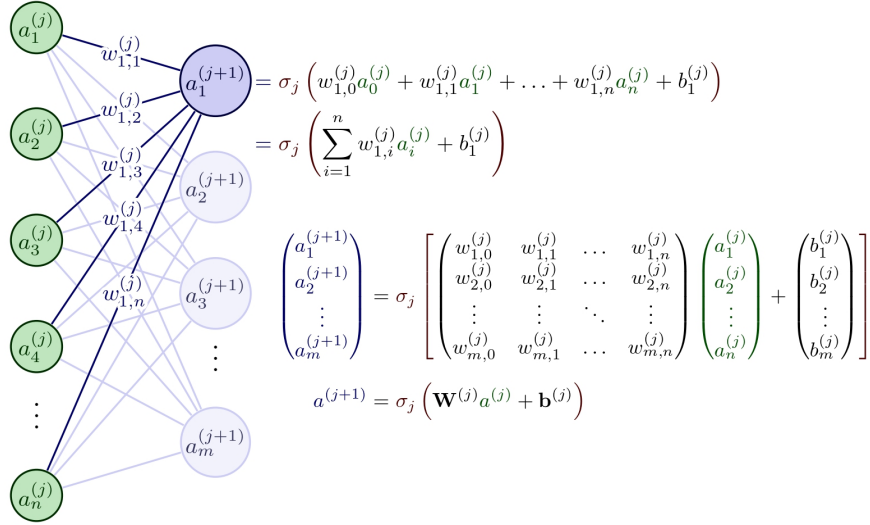


FIGURE 2 – Passage d'une couche à la prochaine couche [1]

Pour la couche de sortie, en fonction des applications on choisit une fonction d'activation σ' différente de σ (par exemple l'identité pour la régression et softmax pour la classification).

1.2 Source d'incertitudes dans un réseaux de neurones

On modélise le réseau de neurones *feedforward* comme la fonction non linéaire

$$f : (x, \theta) \in \mathcal{X} \times \Theta \mapsto f(x, \theta) \in \mathcal{Y}$$

où

- $\mathcal{X} \subset \mathbb{R}^{n_e}$, l'espace des variables d'entrée
- $\mathcal{Y} \subset \mathbb{R}^{n_s}$, l'espace des variables de sortie
- $\Theta \subset \mathbb{R}^{n_p}$, l'espace des paramètres

On se donne maintenant une base de données d'entraînement $D = \{(x_i, y_i)\}_{i=1}^N \in (\mathcal{X} \times \mathcal{Y})^N$.

1.2.1 Acquisition des données

Si on prend comme données d'entrée x une mesure d'une quantité réelle \tilde{x} .

Il y a une variabilité sur la mesure en fonction des circonstances et conditions $\omega \in \Omega$ où la mesure a été effectué en plus de la précision de l'appareil de mesure.

La sortie y peut aussi subir des erreurs de labélisation (pour le cas d'une tâche de classification) ou aussi de mesure si c'est une quantité mesurée ou issue d'une quantité mesurée.

En somme on a une incertitude sur l'entrée $x|\omega \sim p_{x|\omega}$ et $y|\omega \sim p_{y|\omega}$.

S'ajoute à cela, l'incertitude sur x qui peut se propager sur y .

1.2.2 Structure du modèle

Les paramètres θ et donc l'espace Θ est variable en fonction du choix de modèle s .

On a $\theta|D, s \sim p_{\theta|D,s}$

1.3 Prédiction

La prédiction faite à partir d'un réseau entraîné subit alors les incertitudes venant de ces sources. La distribution de la prédiction y^* sachant une entrée x^* est donné par

$$p(y^*|x^*, D) = \int_{\Theta} \underbrace{p(y^*|x^*, \theta)}_{\text{Données}} \underbrace{p(\theta|D)}_{\text{Modèle}} d\theta$$

1.3.1 Incertitude aléatoire

L'incertitude dite aléatoire est due à la variabilité innée aux données d'entraînement du modèle. Elle affecte la partie $p(y^*|x^*, \theta)$ de la tâche de prédiction.

1.3.2 Incertitude épistémique

L'incertitude épistémique affecte la partie $p(\theta|D)$ de la tâche de prédiction, elle est due :

- à la complexité du modèle,
- aux erreurs durant la phase d'entraînement,
- à la manque d'information à cause de données manquantes ou la capacité de représentation des données d'entraînement.

2 Réseau de neurones bayésiens (BNN)

Les réseaux de neurones bayésiens sont des réseaux de neurones dont les paramètres ($\theta = (w, b)$) sont, non pas des quantités déterministes (comme dans le cas d'un réseau classique) mais des distributions de probabilité.

À l'initialisation, les paramètres suivent une loi a priori $p(\theta)$, et l'entraînement consiste à évaluer l'a posteriori de cette loi conditionnée aux données d'entraînement $p(\theta|D)$.

Cette méthode permet de tenir compte de l'incertitude sur le modèle utilisé avec une structure donnée, car ici, il s'agit d'évaluer une famille de modèles au lieu d'un seul.

Cependant, à cause de la taille et de la complexité de ces modèles on ne dispose pas, dans le cas général, d'une formule analytique pour calculer cette distribution a posteriori. Des méthodes numériques ont été développés pour calculer une approximation de cette distribution a posteriori.

Parmi ces méthodes on trouve :

- Méthodes variationnelles
- Méthodes par échantillonnage ou Monte-Carlo (qu'on va voir dans la suite)
- Méthodes de Laplace

2.1 Méthodes variationnelles

Ici on approche $p(\theta|D)$ par une famille de distribution paramétrique $\{q^\gamma(\theta)\}_\gamma$ (souvent des gaussiennes). Le but est de choisir γ qui rapproche $q^\gamma(\theta)$ le plus de $p(\theta|D)$ au sens de la divergence de Kullback-Leibler :

$$KL(q||p) = \mathbb{E}_q \left[\log \frac{q^\gamma(\theta)}{p(\theta|D)} \right]$$

Comme on ne connaît pas $p(\theta|D)$, on utilise l'ELBO (*evidence lower bound*) qui est égal à la divergence à une constante près

$$L = \mathbb{E}_q \left[\log \frac{p(y|D, \theta)}{q^\gamma(\theta)} \right]$$

(on a en effet $KL(q||p) = -L + \log p(y|D)$)

On résout ici un problème d'optimisation.

2.2 Méthode de Laplace

$\hat{\theta}$ l'estimateur de maximum d'a priori.

$$\log p(\theta|D) \approx \log p(\hat{\theta}|D) + \frac{1}{2}(\theta - \hat{\theta})^T (H + \tau I)(\theta - \hat{\theta})$$

$$p(\theta|D) \sim \mathcal{N}(\hat{\theta}, (H + \tau I)^{-1})$$

2.3 Méthodes par échantillonnage ou Monte Carlo

La formule de Bayes nous donne

$$p(\theta|D) = \frac{p(D|\theta)}{p(D)}p(\theta)$$

- $p(D|\theta)$ la vraisemblance des données D sachant le paramètre θ ,
- $p(\theta)$ la distribution a priori de θ ,
- $p(D)$ la distribution des données d'entraînement.

Il s'agit de faire des tirages de θ pour évaluer ces quantités.

3 Algorithmes BNN-ABC-SS

3.1 ABC (*Approximate Bayesian Computation*)

La méthode ABC consiste à évaluer $p(\theta|D)$ sans évaluer la vraisemblance qui peut s'avérer coûteux.

Posons $\hat{y} = f(x, \theta)$ la sortie d'une évaluation de x par réseaux de neurones f avec paramètre θ .

La formule de Bayes nous donne

$$p(\theta, \hat{y}|D) \propto p(D|\hat{y}, \theta)p(\hat{y}|\theta)p(\theta)$$

Pour simuler selon la distribution du second membre, on applique l'algorithme de rejet.

- On tire $\theta \sim p(\theta)$
- On évalue $\hat{y} = f(x, \theta) \sim p(\hat{y}|\theta)$
- On accepte le θ si et seulement si $\hat{y} = y$

Comme \hat{y} est une quantité réelle (a priori à distribution continue), obtenir exactement $\hat{y} = y$ est une condition trop forte pour être atteinte (en un temps raisonnable).

On introduit alors une tolérance ϵ , on remplace $\hat{y} = y$ par $|\hat{y} - y| < \epsilon$.

On remarque que plus ϵ est petit, plus on se rapproche de la condition $\hat{y} = y$, et donc le mieux notre approximation sera.

On note $p_\epsilon(\theta, \hat{y}|D)$ la distribution issue du tirage précédent, et qui approche $p(\theta, \hat{y}|D)$, on a

$$p_\epsilon(\theta, \hat{y}|D) \propto \mathbb{1}_{\mathcal{N}_\epsilon(D)}(\hat{y})p(\hat{y}|\theta)p(\theta)$$

où

$$\mathcal{N}_\epsilon(D) = \{\hat{y} \in \mathcal{Y}, \rho(\eta(\hat{y}), \eta(y)) \leq \epsilon\}$$

où η est une statistique qui caractérise une distribution (par exemple les moments ou les quantiles) et ρ est une mesure de dissimilarité.

En intégrant des deux côtés par \hat{y} on obtient notre approximation de $p(\theta|D)$

$$\begin{aligned} p_\epsilon(\theta|D) &= \int_{\mathcal{Y}} p_\epsilon(\theta, \hat{y}|D) d\hat{y} \propto \int_{\mathcal{Y}} \mathbb{1}_{\mathcal{N}_\epsilon(D)}(\hat{y})p(\hat{y}|\theta)p(\theta) d\hat{y} \\ &= p(\theta) \int_{\mathcal{Y}} \mathbb{1}_{\mathcal{N}_\epsilon(D)}(\hat{y})p(\hat{y}|\theta) d\hat{y} = \mathbb{P}(\hat{y} \in \mathcal{N}_\epsilon(D)|\theta)p(\theta) \end{aligned}$$

Cependant lors de l'algorithme de rejet, tirer les θ de manière générique donne trop rarement des \hat{y} qui tombent dans $\mathcal{N}_\epsilon(D)$ ce qui fait que l'approximation est très lente à converger. On utilise alors SS (Subset Simulation) pour faire des tirages plus fins.

3.2 SS (*Subset Simulation*)

Soient $\epsilon_1 > \epsilon_2 > \dots > \epsilon_m = \epsilon$ des seuils.

Il est clair que $\mathcal{N}_{\epsilon_m}(D) \subset \mathcal{N}_{\epsilon_{m-1}}(D) \subset \dots \subset \mathcal{N}_{\epsilon_2}(D) \subset \mathcal{N}_{\epsilon_1}(D)$.

De plus et en conséquence, $\bigcap_{j=1}^m \mathcal{N}_{\epsilon_j} = \mathcal{N}_{\epsilon_m} = \mathcal{N}_\epsilon$, ce qui nous donne

$$\begin{aligned}
\mathbb{P}(\hat{y} \in \mathcal{N}_\epsilon(D)|\theta) &= \mathbb{P}\left(\hat{y} \in \bigcap_{j=1}^m \mathcal{N}_{\epsilon_j}(D)|\theta\right) \\
&= \mathbb{P}(\hat{y} \in \mathcal{N}_{\epsilon_1}(D)|\theta) \prod_{j=2}^m \mathbb{P}(\hat{y} \in \mathcal{N}_{\epsilon_j}(D)|\hat{y} \in \mathcal{N}_{\epsilon_{j-1}}(D), \theta)
\end{aligned}$$

L'idée de la SS, est de faire des tirages itératifs de plus en plus fins.

Initialement on tire de manière générique avec un ϵ_1 grand.

À chaque itération, on tire à partir (conditionnés) des tirages précédents avec une tolérance ϵ_i plus petite.

Après n itérations, on aura tiré avec ϵ_n beaucoup plus petit que ϵ_0 .

Cette méthode exploite la décomposition d'une probabilité d'un ordre très petit en un produit de probabilité d'un ordre assez grand pour être calculable en un temps raisonnable.

3.3 Pseudo - Code

Pour la génération des chaines de Markov dans l'espace $\mathcal{N}_{\epsilon_j}(D)$, On utilise l'algorithme 'Modified Monte Carlo' [4, 3].

L'algorithme de MCMC [2] appliquées à notre problème s'écrit se déroule comme suit.

On se donne une distribution de proposition $q(.|.)$ et au niveau j , à l'étape n de la chaîne :

- On génère $\theta' \sim q(\theta|\theta)$ et $y' \sim p(x|\theta')$
- On accepte (θ', x') en tant que $(\theta^{(n)}, x^{(n)})$ avec une probabilité :

$$\alpha = \min \left\{ 1, \frac{p(\theta')q(\theta^{(n-1)}|\theta')}{p(\theta^{(n-1)})q(\theta'|\theta^{(n-1)})} \right\} \mathbb{1}_{\mathcal{N}_{\epsilon_j}}(x')$$
- Sinon $(\theta^{(n)}, x^{(n)}) = (\theta^{(n-1)}, x^{(n-1)})$

Dans les exemples qu'on va prendre après, on a pris

- $p(\theta)$ est la densité de la distribution $\mathcal{N}(0, \sigma_0)$
- $q(.|b)$ est celle de $\mathcal{N}(b, \sigma_j)$

Algorithme 1 : Pseudo code BNN - ABC - SS

Entrées : $N \in \mathbb{N}^*$: le nombre de tirages à chaque itération

$l_{\max} \in \mathbb{N}^*$: le nombre d'itérations maximaux

P_0 la proportion de nos tirages qu'on garde pour générer à la prochaine itération

ϵ : la tolérance finale

x : la variable d'entrée

y : la variable de sortie

Sorties : $[\theta_1^{(n)}, \dots, \theta_N^{(n)}]$

$NP_0 \leftarrow N * p_0$;

$iP_0 \leftarrow p_0^{-1}$;

$[\theta_1^{(0)}, \dots, \theta_N^{(0)}]$, N tirage de $\theta \sim p(\theta)$;

$[\hat{y}_1^{(0)}, \dots, \hat{y}_1^{(0)}] \leftarrow [f(x, \theta_1^{(0)}), \dots, f(x, \theta_N^{(0)})]$;

$[\gamma_1^{(0)}, \dots, \gamma_1^{(0)}] \leftarrow [\rho(\eta(y), \eta(\hat{y}_1^{(0)})), \dots, \rho(\eta(y), \eta(\hat{y}_N^{(0)}))]$;

pour $j \in \{1, \dots, l_{\max}\}$ **faire**

 On ordonne $[\gamma_1^{(j-1)}, \dots, \gamma_1^{(j-1)}]$ dans l'ordre croissant ;

 On réordonne $[\theta_1^{(j-1)}, \dots, \theta_N^{(j-1)}]$ dans cet ordre ;

$\epsilon_j \leftarrow \gamma_{NP_0}^{(j-1)}$ (ou $\frac{1}{2}(\gamma_{NP_0}^{(j-1)} + \gamma_{NP_0+1}^{(j-1)})$) ;

pour $k \in \{1, \dots, NP_0\}$ **faire**

 On choisit une graine $\theta_k^{(j-1),1} = \theta_k^{(j-1)}$ tel que $\hat{y}_k^{(j-1)} \in \mathcal{N}_{\epsilon_j}(D)$

 On génère iP_0 états d'une chaîne de Markov de θ tel que $\hat{y} \in \mathcal{N}_{\epsilon_j}(D)$:

$[\theta_k^{(j-1),1}, \dots, \theta_k^{(j-1),iP_0}]$ et avec ça $[\gamma_k^{(j-1),1}, \dots, \gamma_k^{(j-1),iP_0}]$

fin

$[\theta_1^{(j)}, \dots, \theta_N^{(j)}] \leftarrow [\theta_k^{(j-1),l}, k \in \{1, \dots, NP_0\}, l \in \{1, \dots, iP_0\}]$

$[\gamma_1^{(j)}, \dots, \gamma_N^{(j)}] \leftarrow [\gamma_k^{(j-1),l}, k \in \{1, \dots, NP_0\}, l \in \{1, \dots, iP_0\}]$

si $\epsilon_j \leq \epsilon$ **alors**

 Fin de l'algorithme ;

fin

fin

4 Réalisation

On se donne une base de données à étudier avec une comparaison avec des méthodes déjà existante [4, 5, 6] Ici on propose de reproduire un résultat vu dans l'article [5].

4.1 Cosinus perturbé

Les données d'entraînement $D = \{x_i, y(x_i)\}_{i=1}^{200}$, avec $(x_i)_{i=1}^N$ une discrétisation uniforme de $[-3, 3]$ et $y(x) = \cos(x) + \zeta$ où $\zeta \sim \mathcal{N}(0, 0.1)$.

Les paramètres de l'algorithme $N : 5000$, $l_{max} : 6$, $P_0 : 0.2$ et $\epsilon : 0.1$.

Le réseau de neurones pris ici a une couche d'entrée, une couche cachée à deux cellules et une couche de sortie ce qui fait 7 paramètres avec la fonction sigmoïde comme fonction d'activation. La mesure de dissimilarité choisi est la MSE (*Mean Squared Error*).

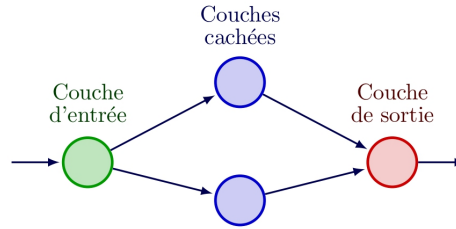


FIGURE 3 – Réseau de neurones pour la fonction cos

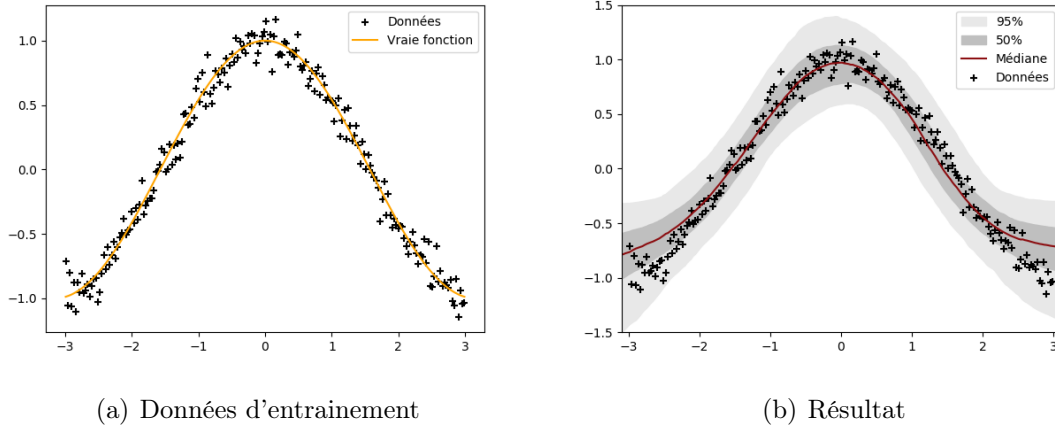


FIGURE 4 – Résultat de l'algorithme BNN-ABC-SS

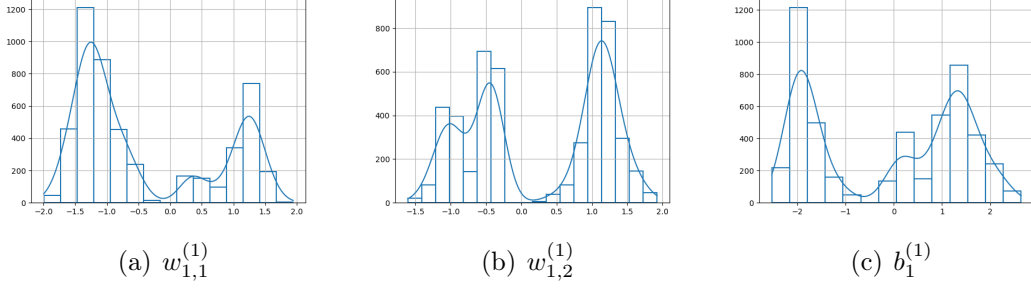


FIGURE 5 – Exemple de distribution a posteriori des poids et biais

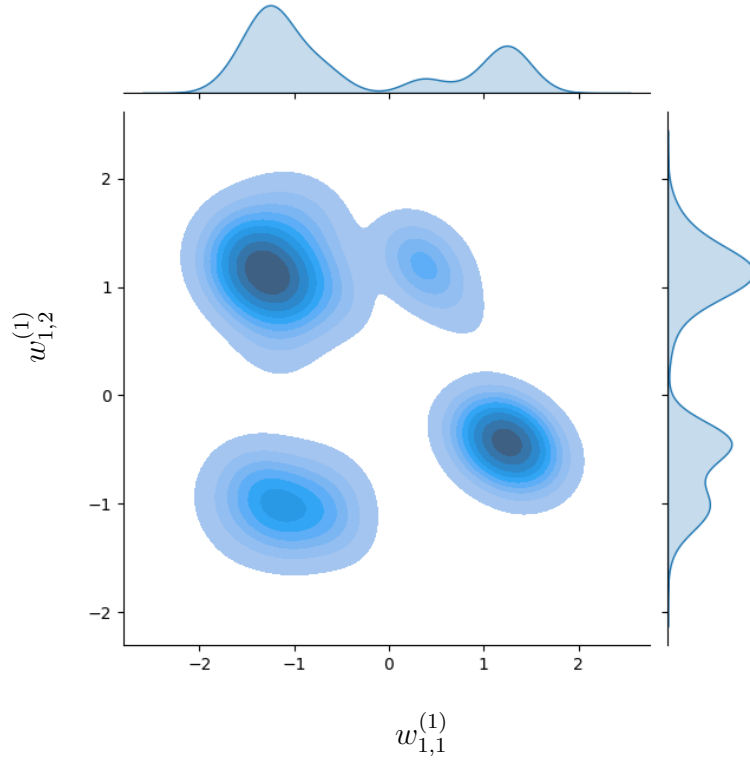


FIGURE 6 – Distribution a posteriori de $w_{1,1}^{(1)}$ et $w_{1,2}^{(1)}$

4.2 Sinus perturbé

Les données d'entraînement $D = \{x_i, y(x_i)\}_{i=1}^{100}$, avec $(x_i)_{i=1}^N$ une discrétisation uniforme de $[-0.5, 0.5]$ et $y(x) = 10 \sin(2\pi x) + \zeta$ où $\zeta \sim \mathcal{N}(0.1)$. Les paramètres de l'algorithme $N : 20000$, $l_{max} : 20$, $P_0 : 0.1$ et $\epsilon : 0.1$.

Le réseau de neurones pris ici a une couche d'entrée, deux couches cachées à 15 cellules et une couche de sortie ce qui fait 286 paramètres avec la fonction ReLu comme fonction d'activation. La mesure de dissimilarité choisie est la MSE (*Mean Squared Error*).

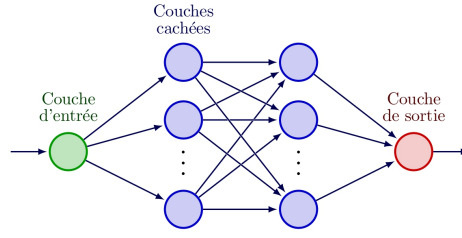


FIGURE 7 – Réseau de neurones pour la fonction sin

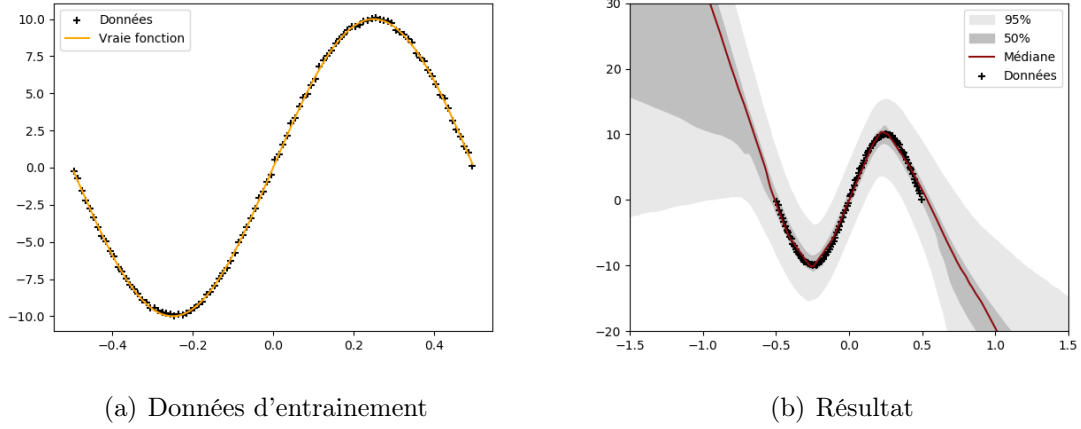


FIGURE 8 – Résultat de l'algorithme BNN-ABC-SS

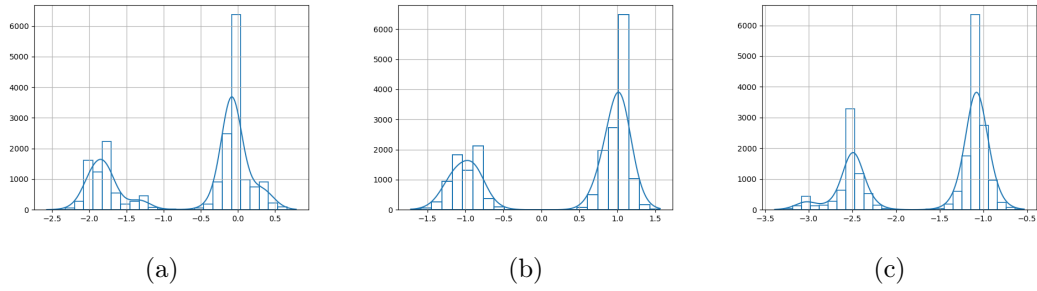


FIGURE 9 – Exemple de distribution a posteriori des poids et biais

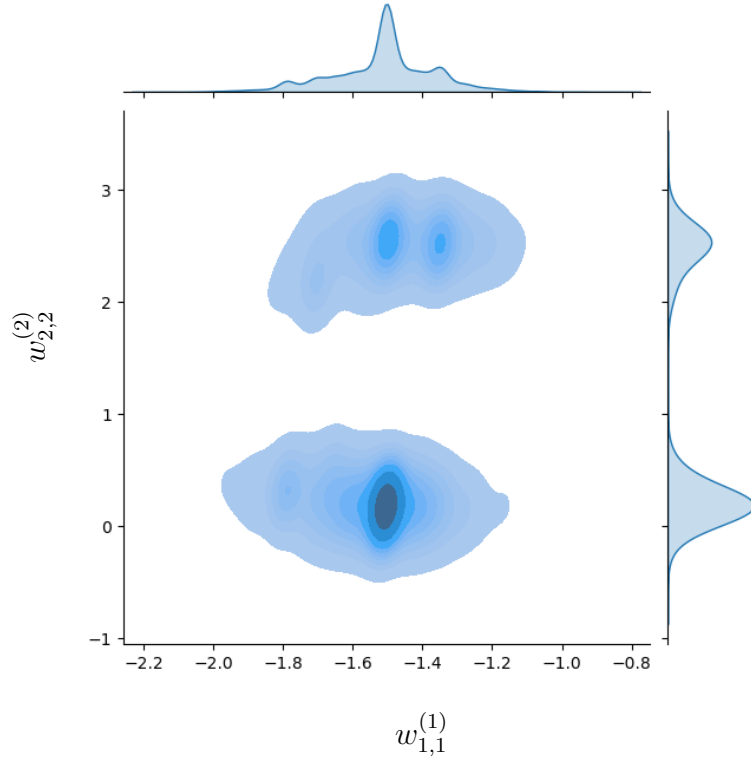


FIGURE 10 – Distribution a posteriori de $w_{1,1}^{(1)}$ et $w_{2,2}^{(2)}$

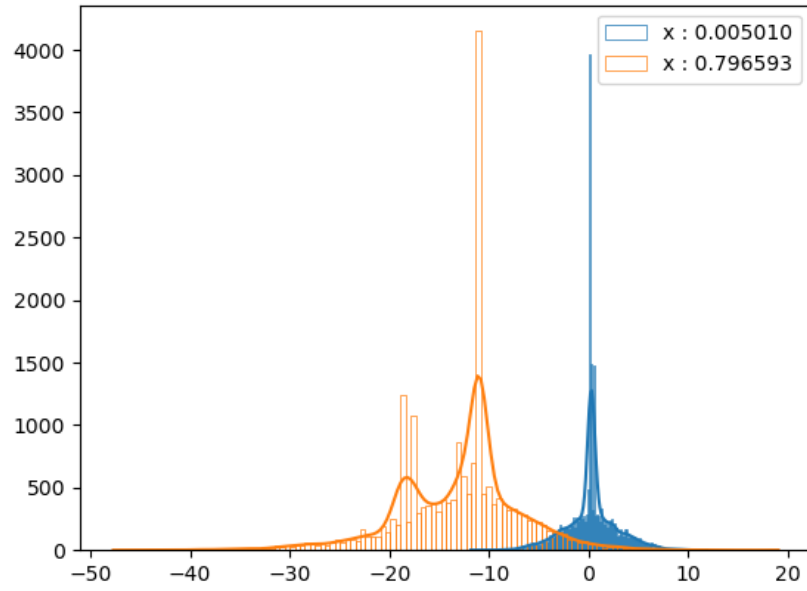


FIGURE 11 – Distribution sur la sortie pour $x \approx 0$ dans le domaine et $x \approx 0.8$ hors du domaine

4.3 Maybe more

Insert im here
list hypperamettters
Insert res here

5 Conclusion et perspective

On a vu ici l'implémentation d'une méthode d'approximation sur l'incertitude sur la sortie qui pourra être utilisée par la suite dans

Références

- [1] https://tikz.net/neural_networks/.
- [2] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to mcmc for machine learning. *Machine Learning*, 50(1/2) :5–43, 2003.
- [3] Siu-Kui Au and James L. Beck. Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic Engineering Mechanics*, 16(4) :263–277, October 2001.
- [4] Manuel Chiachio, James L. Beck, Juan Chiachio, and Guillermo Rus. Approximate bayesian computation by subset simulation. *SIAM Journal on Scientific Computing*, 36(3) :A1339–A1358, January 2014.
- [5] Juan Fernández, Manuel Chiachío, Juan Chiachío, Rafael Muñoz, and Francisco Herrera. Uncertainty quantification in neural networks by approximate bayesian computation : Application to fatigue in composite materials. *Engineering Applications of Artificial Intelligence*, 107 :104511, January 2022.
- [6] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Hunt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, Muhammad Shahzad, Wen Yang, Richard Bamler, and Xiao Xiang Zhu. A survey of uncertainty in deep neural networks, 2021.