PFE

Khaled MEDJKOUH
Davidson Lova RAZAFINDRAKOTO

June 6, 2023

Contents

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'entrainement. En fonction des conditions de l'entrainement du modèles (données d'entrainement, 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 [6], 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.

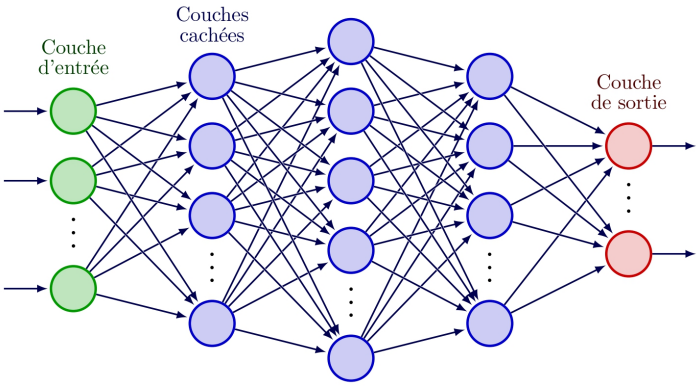


Figure 1: Schéma de réseaux *feedforward* [1]

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.

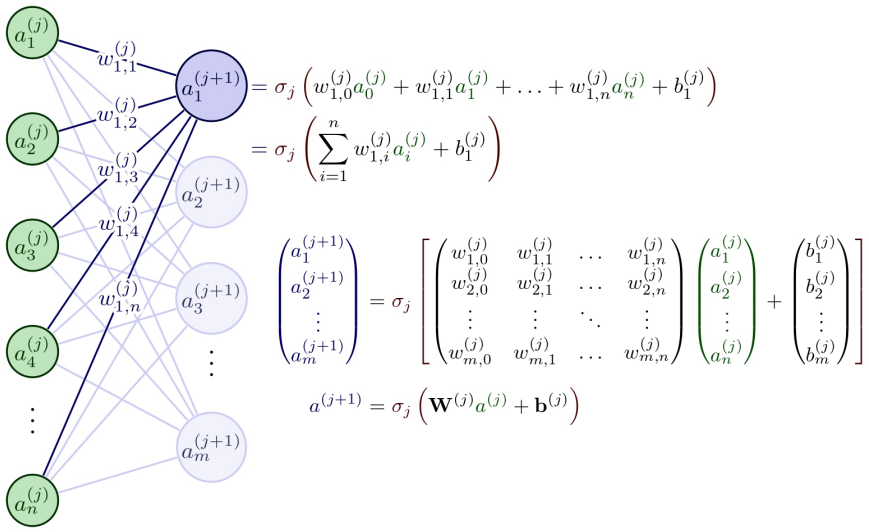


Figure 2: Passage d'une couche à la prochaine couche [1]

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

$$f : (x, \theta) \in \mathcal{X} \times \Theta \mapsto f(x, \theta) \in \mathcal{Y} \tag{1.2.1}$$

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'entrainement $D = \{(x_i, y_i)\}_{i=1}^{N} \in (\mathcal{X} \times \mathcal{Y})^N$.

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 entrainé subit alors les incertitudes venant de ces sources. La distribution de la prédiction y^* sachant une entrée x^* est donné par

$$p(y^*|x^*, D) = \int_\Theta \underbrace{p(y^*|x^*, \theta)}_{\text{Données}} \underbrace{p(\theta|D)}_{\text{Modèle}} d\theta \tag{1.3.2}$$

1.3.1 Incertitude aléatoire

L'incertitude dite aléatoire est due à la variabilité innée aux données d'entrainement 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'entrainement,
- à la manque d'information à cause de données manquantes ou la capacité de représentation des données d'entrainement.

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'entrainement consiste à évaluer l'a posteriori de cette loi conditionnée aux données d'entrainement $p(\theta|D)$.

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.

Parmis 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

On approche $p(\theta|D)$ en choisissant une distribution parmi une famille de distributions paramétrées $\{q^\gamma(\theta)\}_\gamma$ (souvent des gaussiennes). Le but est de choisir la distribution $q^\gamma(\theta)$ qui se rapproche le plus de $p(\theta|D)$ au sens de la divergence de Kullback-Leibler :

$$KL(q^\gamma||p) = \mathbb{E}_q\left[\log \frac{q^\gamma(\theta)}{p(\theta|D)}\right] \tag{2.1.3}$$

Comme une distribution est determinée par leur paramètre, le choix de la distribution optimale q^* est réduit au choix du paramètre optimal γ^*.

$$\gamma^* = \arg\min_{\gamma\in\Gamma} KL(q^\gamma||p) \tag{2.1.4}$$

2.2 Méthode de Laplace

On approche la distribution a priori par une gaussienne centrée autour de $\hat{\theta}$ l'estimateur de maximum d'a priori avec une variance issue du résultat du développement limité ci-dessous [7] .

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

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

Ici H est la matrice hessienne de $log(p(\theta|D))$.

2.3 Méthodes par échantillonage 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'entrainement.

Si on arrive à evaluer la vraisemblance, et comme $p(\theta|D) \propto p(D|\theta)p(\theta)$, on peut tracer une chaine de Markov qui simulent des échantillons de $p(\theta|D)$ en utilisant la vraisemblance et la distribution a priori. Un algorithme qui permet de générer cette chaine de Markov est par exemple l'algorithme de Metropolis Hastings [7].

Après avoir générer des échantillons $p(\theta|D)$, on estime $p(\theta|D)$ à l'aide de ces échantillons.

3 Algorithme BNN-ABC-SS

3.1 ABC (*Approximate Bayesian Computation*)

On se donne une donnée $D = (x, y)$.

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

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

$$p(\theta, \hat{y}|D) \propto p(D|\hat{y}, \theta)p(\hat{y}|\theta)p(\theta) \tag{3.1.5}$$

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

- On tire $\theta \sim p(\theta)$
- On evalue $\hat{y} = f(x, \theta) \sim p(\hat{y}|\theta)$
- 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

$$p_\epsilon(\theta, \hat{y}|D) \propto \mathbb{1}_{\mathcal{N}_\epsilon(D)}(\hat{y})p(\hat{y}|\theta)p(\theta) \tag{3.1.6}$$

où

$$\mathcal{N}_\epsilon(D) = \{\hat{y} \in \mathcal{Y}, \rho(\eta(\hat{y}), \eta(y)) \leq \epsilon\} \tag{3.1.7}$$

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 $p(\theta|D)$

$$p_\epsilon(\theta|D) = \int_{\mathcal{Y}} p_\epsilon(\theta, \hat{y}|D)d\hat{y} \propto \int_{\mathcal{Y}} \mathbb{I}_{\mathcal{N}_\epsilon(D)}(\hat{y})p(\hat{y}|\theta)p(\theta)d\hat{y} \tag{3.1.8}$$

$$= p(\theta) \int_{\mathcal{Y}} \mathbb{I}_{\mathcal{N}_\epsilon(D)}(\hat{y})p(\hat{y}|\theta)d\hat{y} = \mathbb{P}(\hat{y} \in \mathcal{N}_\epsilon(D)|\theta)p(\theta) \tag{3.1.9}$$

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 prend beaucoup de temps à converger. On utilise alors SS (Subset Simulation) pour faire des tirages plus fins.

3.2 SS (*Subset Simulation*)

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

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

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

$$\mathbb{P}\left(\hat{y} \in \mathcal{N}_{\epsilon}(D)|\theta\right) = \mathbb{P}\left(\hat{y} \in \bigcap_{j=1}^{m} \mathcal{N}_{\epsilon_j}(D)|\theta\right) \tag{3.2.10}$$

$$= \mathbb{P}\left(\hat{y} \in \mathcal{N}_{\epsilon_1}(D)|\theta\right) \prod_{j=2}^{m} \mathbb{P}\left(\hat{y} \in \mathcal{N}_{\epsilon_j}(D)|\hat{y} \in \mathcal{N}_{\epsilon_{j-1}}(D), \theta\right) \tag{3.2.11}$$

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_m 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' [5, 4].

L'algorithme de MCMC [3] 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 chaine :

- On génère $\theta' \sim q(\theta'|\theta)$ et $y' \sim p(y'|x, \theta')$
- On accepte (θ', y') en tant que $(\theta^{(n)}, x^{(n)})$ avec une probabilité :
  $\alpha = \min\left\{1, \frac{p(\theta')\mathbb{1}_{\mathcal{N}_{\epsilon_j}}(y')q(\theta^{(n-1)}|\theta')}{p(\theta^{(n-1)})q(\theta'|\theta^{(n-1)})}\right\}$
- Sinon $(\theta^{(n)}, x^{(n)}) = (\theta^{(n-1)}, x^{(n-1)})$

Cependant cet algorithme s'avère trop lent car comme les θ vivent dans un espace de grandes dimensions, les probabilités calculées ici sont trop petites. Ce qui motive l'utilisation d'une version modifiée du MCMC [4], qui au lieu d'évaluer des probabilités sur tout le vecteur θ, le fait composante par composante.

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 maximales
P_0 la proportion de nos tirages qu'on grade pour générer à la prochaine itération
ϵ : la tolérance finale
x : la variable d'entrée
y : la variable de sortie

Sorties : $\left[\theta_1^{(n)}, ..., \theta_N^{(n)}\right]$

$NP_0 \leftarrow N * p_0$;
$iP_0 \leftarrow p_0^{-1}$;
$\left[\theta_1^{(0)}, ..., \theta_N^{(0)}\right]$, N tirage de $\theta \sim p(\theta)$;
$\left[\hat{y}_1^{(0)}, ..., \hat{y}_1^{(0)}\right] \leftarrow \left[f(x, \theta_1^{(0)}), ..., f(x, \theta_N^{(0)})\right]$;
$\left[\gamma_1^{(0)}, ..., \gamma_1^{(0)}\right] \leftarrow \left[\rho(\eta(y), \eta(\hat{y}_1^{(0)})), ..., \rho(\eta(y), \eta(\hat{y}_N^{(0)}))\right]$;
pour $j \in \{1, ..., l_{max}\}$ **faire**
  On ordonne $\left[\gamma_1^{(j-1)}, ..., \gamma_1^{(j-1)}\right]$ dans l'ordre croissant;
  On réordonne $\left[\theta_1^{(j-1)}, ..., \theta_N^{(j-1)}\right]$ 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 $k \in \{1, ..., NP_0\}$ **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 chaine de Markov de θ tel que $\hat{y} \in \mathcal{N}_{\epsilon_j}(D)$:
    $\left[\theta_k^{(j-1),1}, ..., \theta_k^{(j-1),iP_0}\right]$ et avec ça $\left[\gamma_k^{(j-1),1}, ..., \gamma_k^{(j-1),iP_0}\right]$
  fin
  $[\theta_1^{(j)}, ..., \theta_N^{(j)}] \leftarrow [\theta_k^{(j-1),l}, k \in \{1, ..., NP_0\}, l \in \{1, ..., iP_0\}]$
  $[\gamma_1^{(j)}, ..., \gamma_N^{(j)}] \leftarrow [\gamma_k^{(j-1),l}, k \in \{1, ..., NP_0\}, l \in \{1, ..., iP_0\}]$
  si $\epsilon_j \leq \epsilon$ **alors**
    Fin de l'algorithme;
  fin
fin

4 Réalisations

On se donne une base de données à étudier avec une comparaison avec des méthodes déjà existantes [5, 6, 7] Ici on propose de reproduire un résultat vu dans l'article [6].

4.1 Cosinus perturbé

Les données d'entrainement $D = \{x_i, y(x_i)\}_{i=1}^{200}$, avec $(x_i)_{i=1}^{200}$ 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 utilisé 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 sgmoïde comme fonction d'activation. La mesure de dissimilarité choisi est la MSE (*Mean Squared Error*).

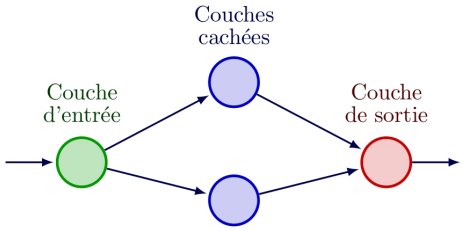
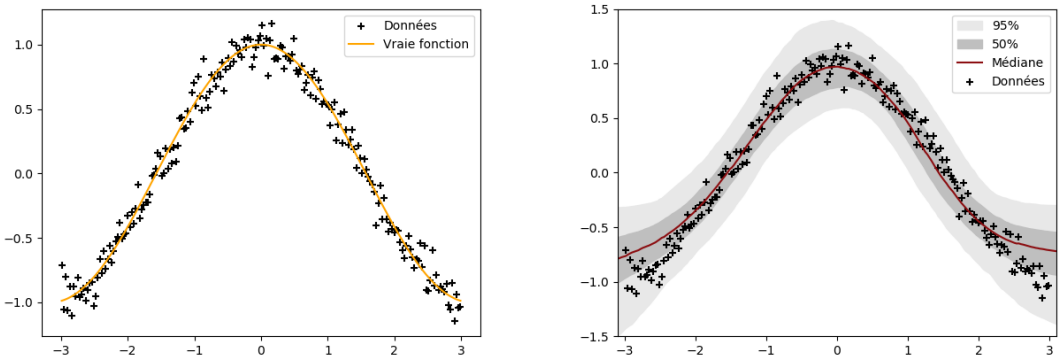


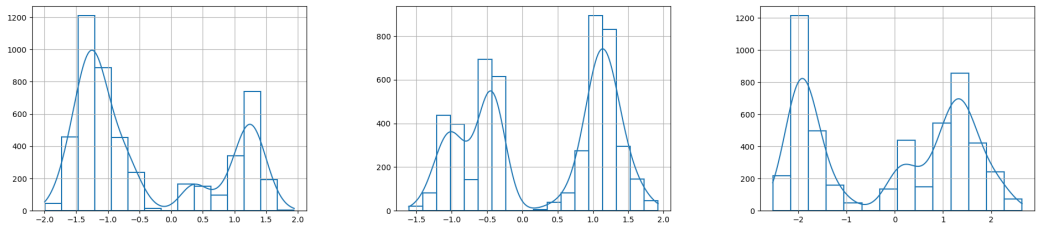
Figure 3: Réseau de neurones pour la fonction cos



(a) Données d'entrainement

(b) Résultat

Figure 4: Résultat de l'algorithme BNN-ABC-SS



(a) $w_{1,1}^{(1)}$ (b) $w_{1,2}^{(1)}$ (c) $b_1^{(1)}$

Figure 5: Exemple de distribution a posteriori des poids et biais

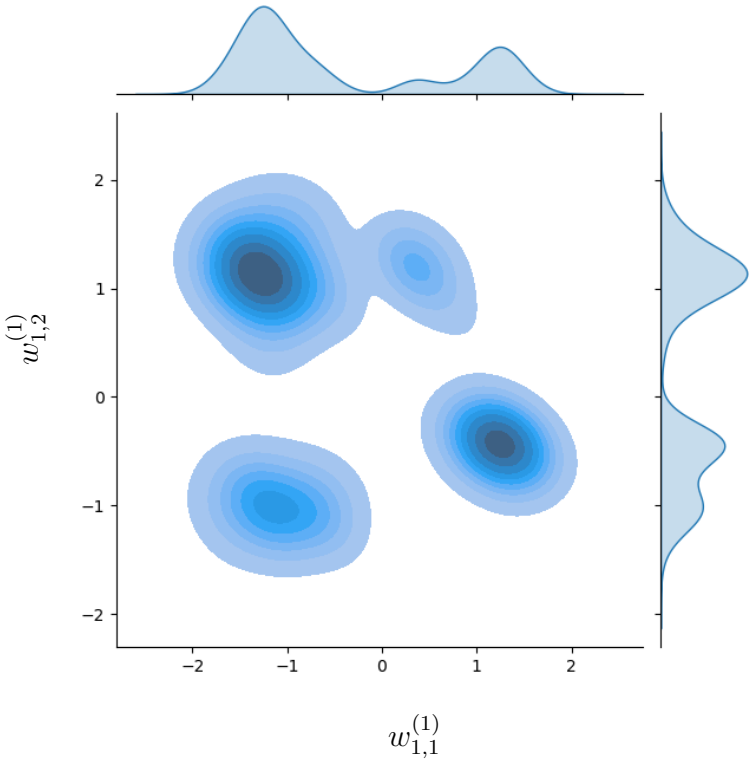


Figure 6: Distribution a posteriori de $w_{1,1}^{(1)}$ et $w_{1,2}^{(1)}$

4.2 Sinus perturbé

Les données d'entrainement $D = \{x_i, y(x_i)\}_{i=1}^{100}$, avec $(x_i)_{i=1}^{100}$ 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*).

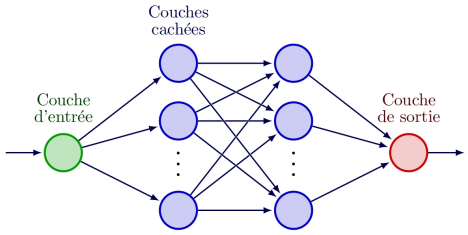
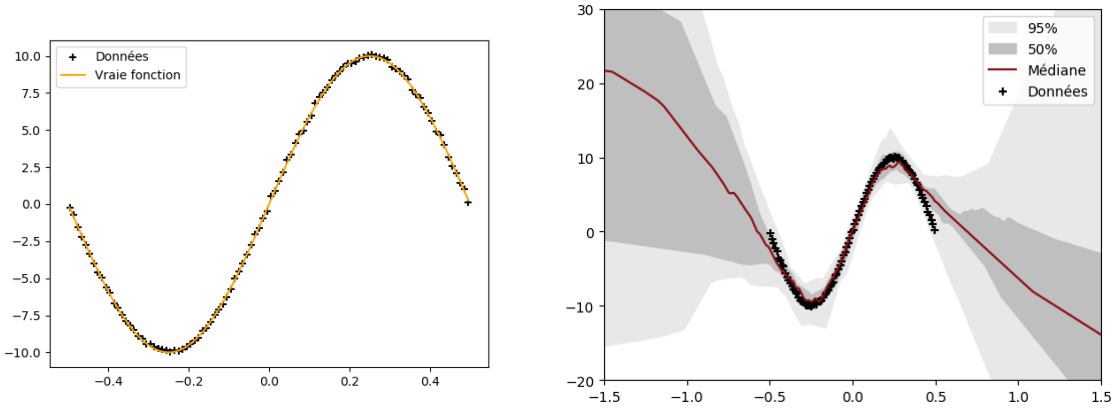
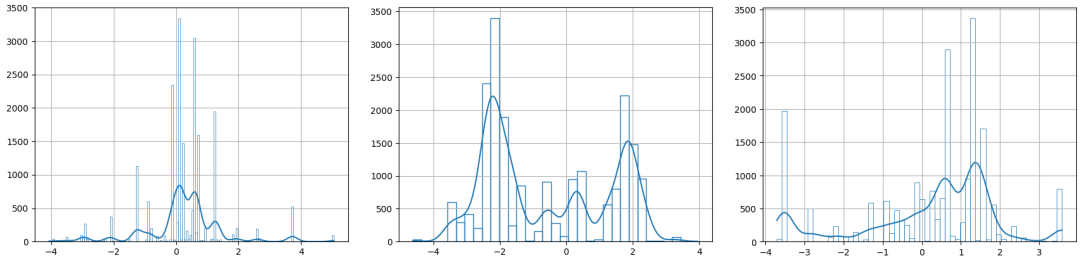


Figure 7: Réseau de neurones pour la fonction sin



(a) Données d'entrainement (b) Résultat

Figure 8: Résultat de l'algorithme BNN-ABC-SS



(a) (b) (c)

Figure 9: Exemple de distribution a posteriori des poids et biais

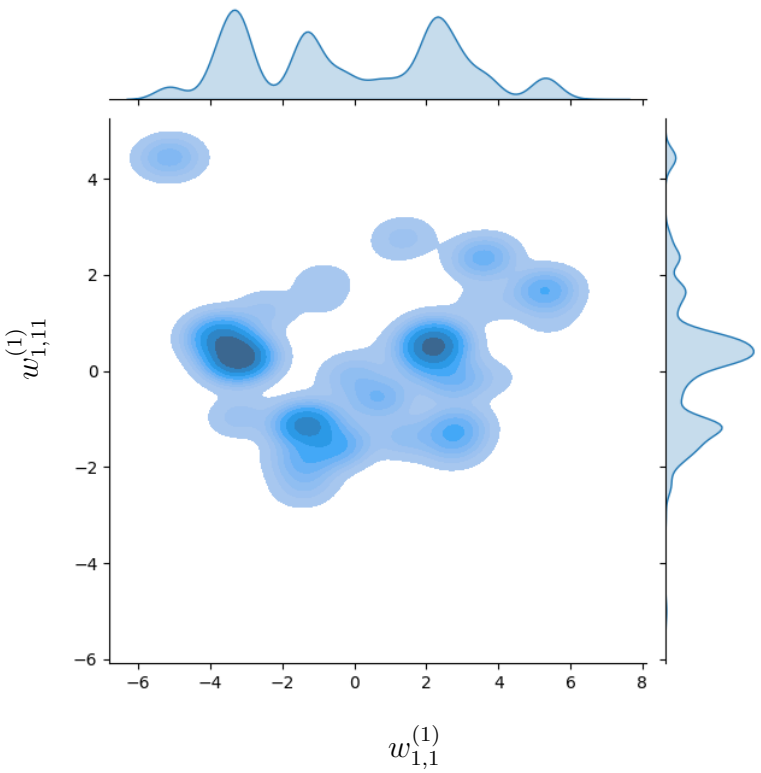


Figure 10: Distribution a posteriori de $w_{1,1}^{(1)}$ et $w_{1,11}^{(1)}$

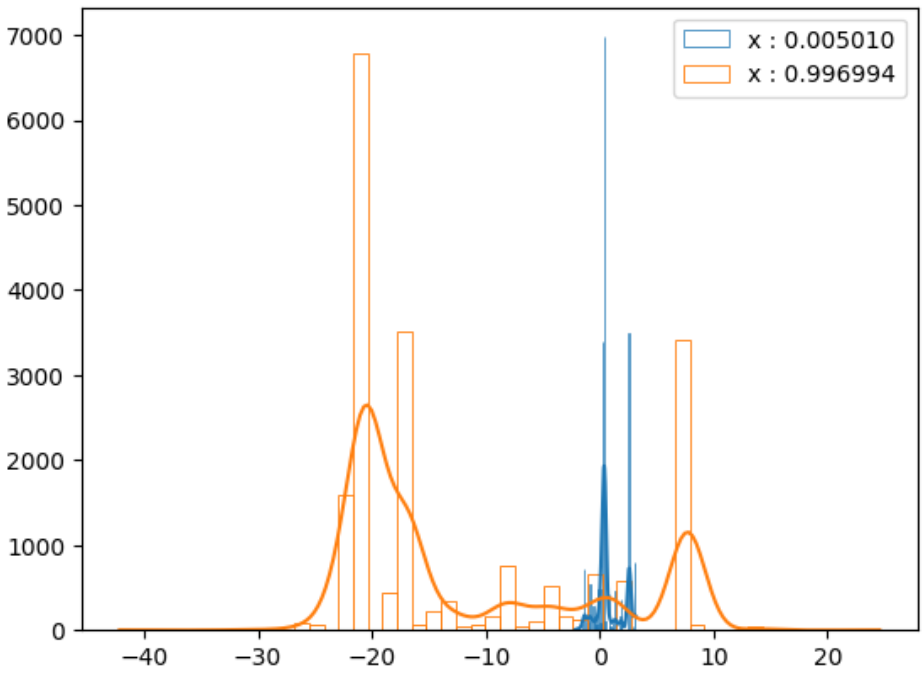


Figure 11: Distribution sur la sortie pour $x \approx 0$ dans le domaine et et $x \approx 1$ hors du domaine

4.3 Application

On utilise ici les données du vol 0 du jeu de données "Aircraft_01.h5".

On souhaite prédire le N2_1 [% rpm] (une description ici) en fonction des variables

- N1_1 [% rpm] (une description ici)
- T1_1 [deg C] (une description ici)
- ALT [ft] (une description ici)
- M [Mach] (une description ici)

On essaie d'abord de faire une régression linéaire i.e. :

$$\forall i, y_i = a_0 + a_1 X_{i,1} + a_2 X_{i,2} + a_3 X_{i,3} + a_4 X_{i,4} + \epsilon_i \Leftrightarrow y = Xa + \epsilon \tag{4.3.12}$$

- y_i la i-ème composante du vecteur de sortie,
- $X_{i,p}$ la i-ème composante de la p-ième covariable,
- ϵ_i le bruit.

avec $X = \begin{pmatrix} 1 & X_{1,1} & X_{1,2} & X_{1,3} & X_{1,4} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{n,1} & X_{n,2} & X_{n,3} & X_{n,4} \end{pmatrix}$, $a = (a_0, ..., a_4)^T$ et $\epsilon = (\epsilon_1, ..., \epsilon_n)^T$

Le paramètre $\hat{a}$ qui minimise l'erreur quadratique moyenne est $\hat{a} = (X^T X)^{-1} X^T y$ et on pose $\hat{y} = X\hat{a}$.

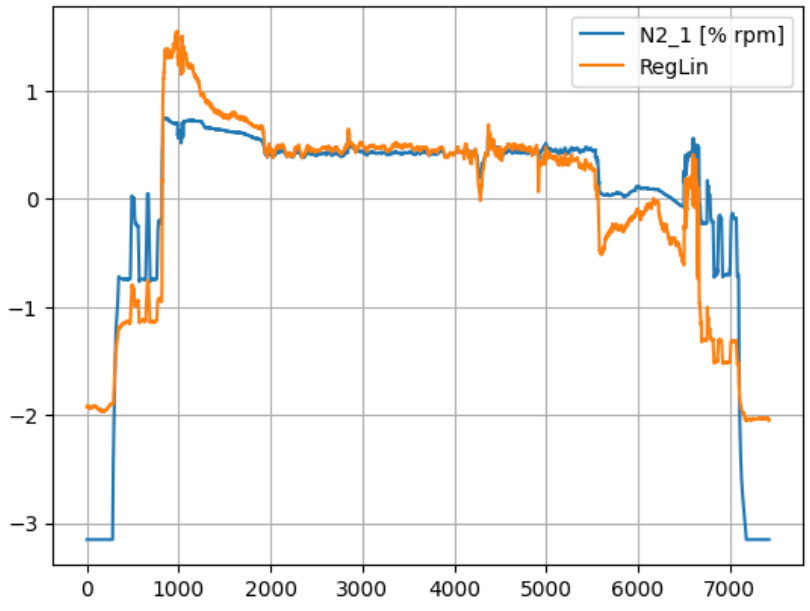
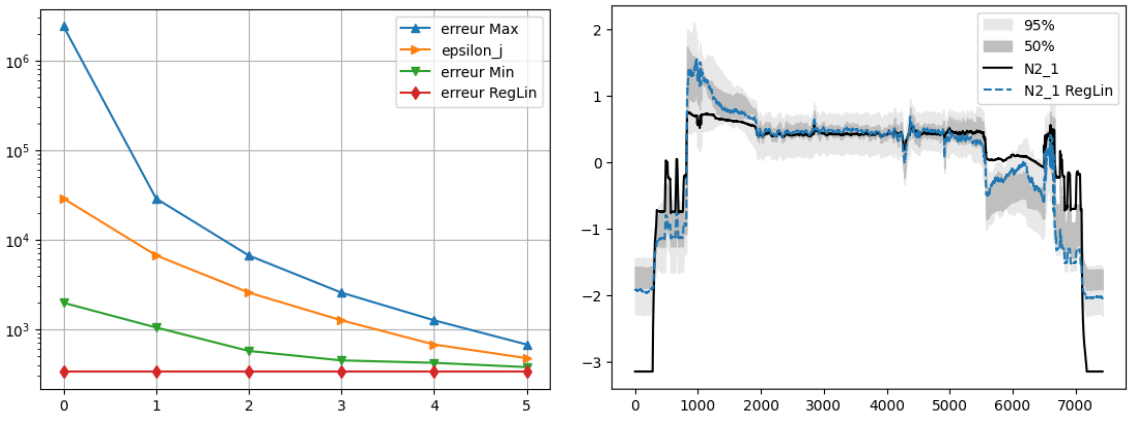


Figure 12: Résultat de la régression linéaire

Maintenant on utilise l'algorithme BNN-ABC-SS avec un modèle linéaire.

Les paramètres de l'algorithme : N : 2000, lmax : 6, P0 : 0.1, epsilon : 0.01 la mesure de dissimilarité choisie est l'erreur quadratique moyenne (MSE).



(a) Courbe d'évolution des erreurs (b) Tube

Figure 13: Résultat de BNN-ABC-SS pour le modèle linéaire

Ensuite on utilise comme modèle un réseau de neurones à une couche cachée composée de 2 neurones avec la fonction d'activation ReLu.

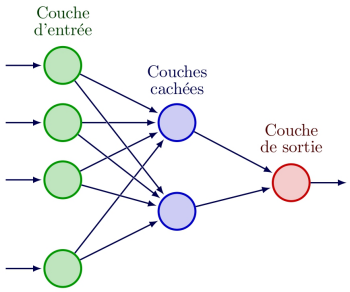
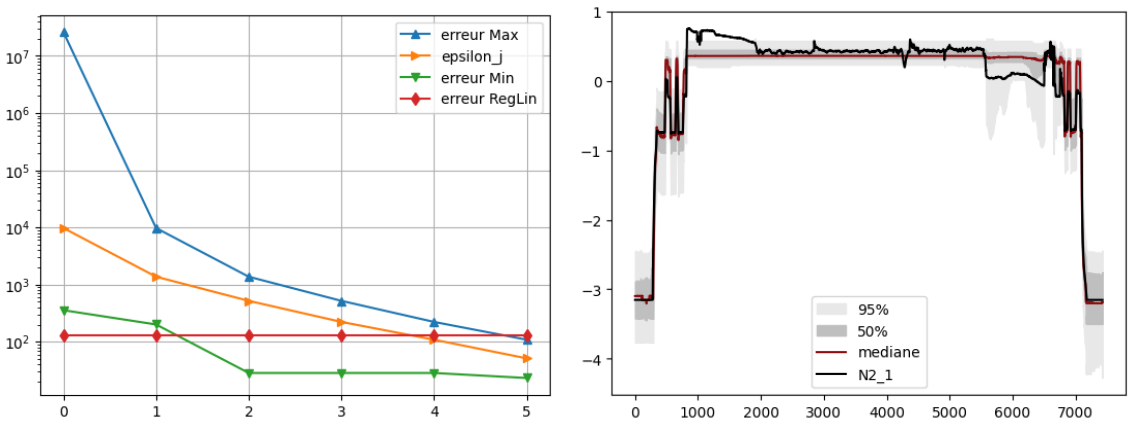


Figure 14: Réseau de neurones utilisé

Les paramètres de l'algorithme : N : 5000, lmax : 6, P0 : 0.1, epsilon : 0.01 et la mesure de dissimilarité choisie est l'erreur quadratique moyenne.

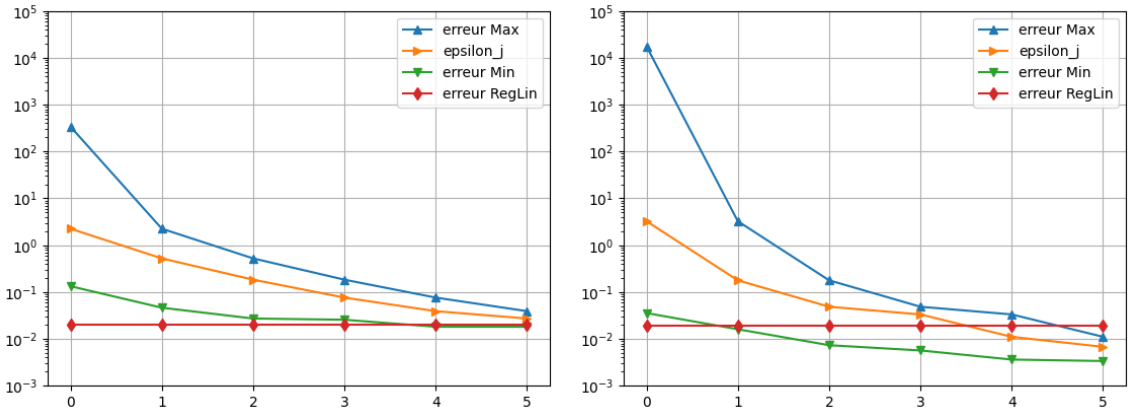


(a) Courbe d'évolution des erreurs (b) Tube

Figure 15: Résultat de BNN-ABC-SS pour le réseau de neurones

Maintenant qu'on a vu qu'on peut appliquées le modèle de cette manière, on va essayer de faire une inférence. On se donne une base de données 10 vols qu'on normalise pour ramener les valeurs dans $[0, 1]$, et on la sépare en 2 échantillons, un d'entrainement et un de test.

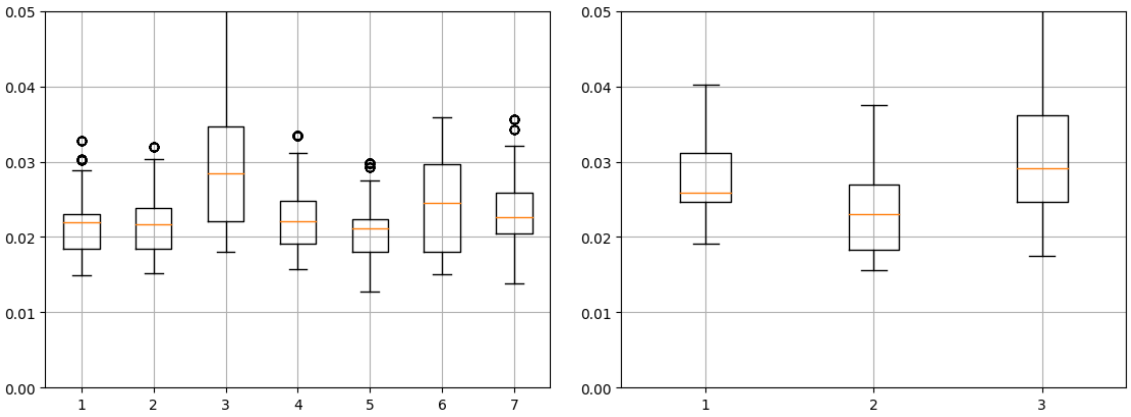
On entraine le modèle (linéaire ensuite le réseau de neurone) sur l'échantillon d'entrainement.



(a) Modèle linéaire (b) réseau de neurones

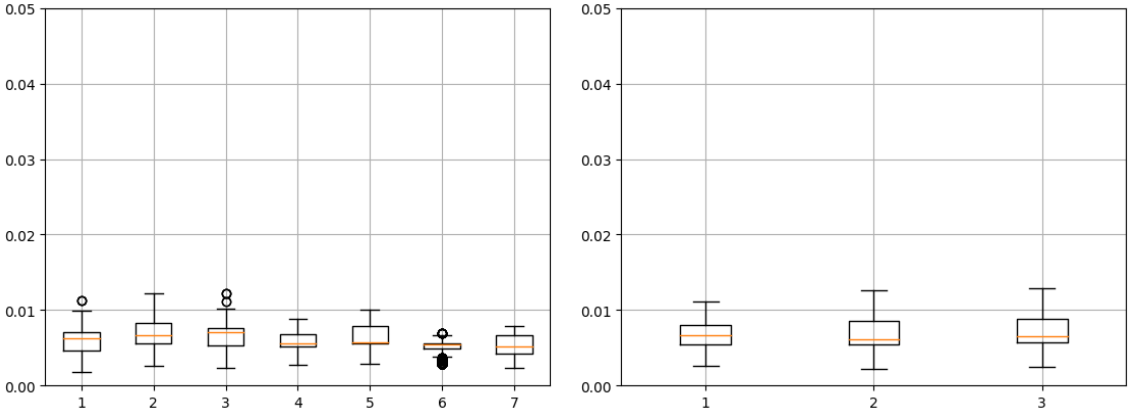
Figure 16:

On obtient alors un modèle moyen (global) sur l'échantillon d'entrainement qu'on évalue pour chaque vol dans ce même échantillion et ensuite pour ceux de test.



(a) Entrainement (b) Test

Figure 17: Erreur MSE pour chaque vol pour le modèle linéaire



(a) Entrainement (b) Test

Figure 18: Erreur MSE pour chaque vol pour le réseau de neurones

5 Conclusion et perspective

Ce travail nous a permis d'implementer un algorithme qui tente d'intégrer un type d'incertitude (épistémique) dans la prédiction d'un réseau de neurones.

Il faut tout de même noter que les résultats obtenus sont fortement sensible au hypeparamètres. Pour améliorer l'algorithme, il faudrait intégrer une phase de sélection de modèle pour :

- fixer la taille minimale de la population de paramètre θ en fonction de la structure du modèle.
- fixer la proportion de population gardée à chaque itération P_0
- fixer le nombre d'itérations maximales l_{max}
- proposer une bonne distribution a priori de θ [3] (sachant que ce choix peut faire la différence entre converger ou pas, pour un a priori gaussien, estimer le bon σ_0)

Pour pousser encore plus loin, il faudrait pouvoir adaptativement, changer la fonction de proposition pour l'algorithme MCMC [3, 5](pour le cas gaussien, fixer σ_j adapativement).

L'ensemble des réalisations sont disponibles sur ce lien github [2].

References

[1] https://tikz.net/neural_networks/.

[2] https://github.com/Davidson-Lova/PFE_clean.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, 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.