PFE

Khaled MEDJKOUH
Davidson Lova RAZAFINDRAKOTO

11 avril 2023

Table des matières

1 Introduction

1.1 Réseau de neurones

Un réseaux des neurones est un fonction qui prends ces arguments dans la couche de sorties et qui sort un résultat dans la couche d'entrée. Chaque couche est composé de cellule contenant des valeurs numériques. Un réseau de neurones Feed-Forward prends l'information dans la couche d'entrée, ensuite chaque couche applique une transformation à cette information jusqu'à la couche finale.

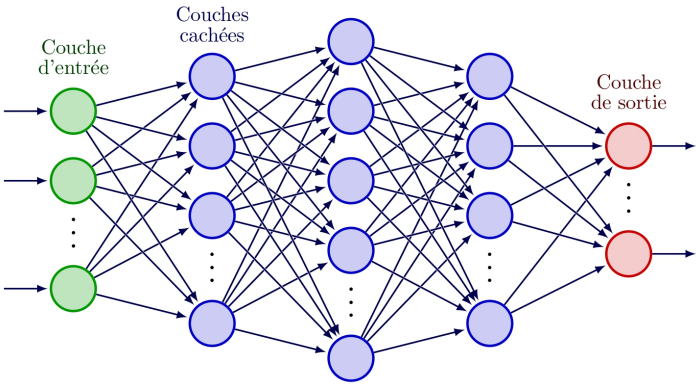


FIGURE 1 – Schéma de réseaux

La couche d'entrée prend la donnée d'entrée x, et la fait passer le long des couches cachées pour finalement arrivée à la couche de sortie. Ci-dessous on voit comment sont évaluer les valeurs pour chaque couche.

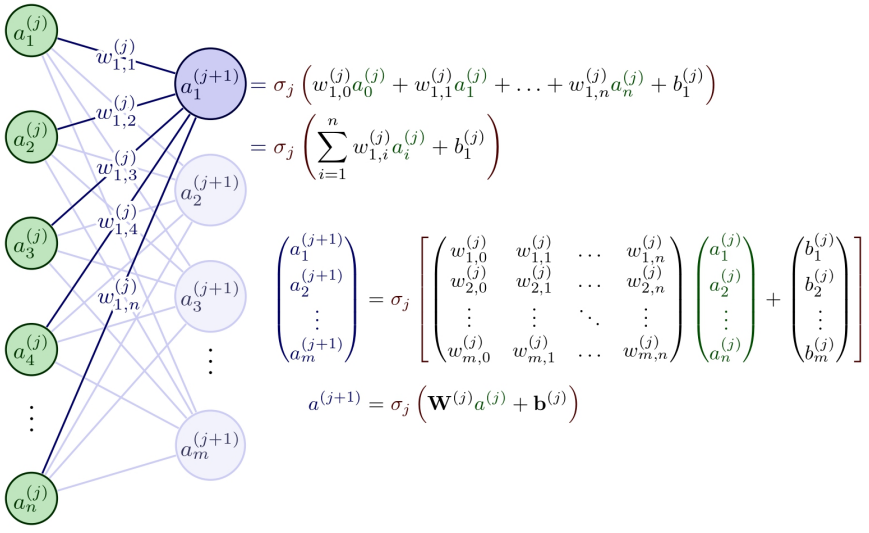


FIGURE 2 – Feed Forward

Ces réseaux

1.2 Source d'incertitudes dans un réseaux de neurones

On modélise le réseaux de neurones 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'entrées

— $\mathcal{Y} \subset \mathbb{R}^{n_s}$, l'espace des variables de sorties

— $\Theta \subset \mathbb{R}^{n_p}$, 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'entrées x une mesure d'une quantité réel $\tilde{x}$.

Il y a une variabilité sur la mesure en fonction des circonstances et conditions $\omega \in \Omega$ dans lequel la mesure a été effectué.

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

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 ça, l'incertitude sur x qui peut se propager sur le y.

1.2.2 Structure du model

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

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

1.3 Prédiction

La distribution de la prédiction y^* sachant une entrée x^* est données 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

Insert definition here

Elle affecte la partie $p(y^*|x^*, \theta)$ de la tâche de prédiction, elle est dus à la variabilité(précision et erreurs) lors des mesures. Elle est donnée et inhérent au problème.

1.3.2 Incertitude épistemique

Insert definition here

Affecte la partie $p(\theta|D)$ de la tâche de prédiction, elle due :

— à la compléxité 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 donnés d'entrainement.

2 Réseau de neurones bayésiens (BNN)

Les réseaux de neurones bayésiens sont des réseaux de neurones dont les poids sont, non pas des quantités déterministes (comme dans le cas d'un NN normale) mais des distributions.

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

On ne dispose pas dans le cas générale, d'une formule analytique de cette distribution a posteriori.

Voici trois familles de méthodes pour approcher cette distribution :

— Méthodes variationelle
— Méthodes par échantillonage ou Monte Carlo (qu'on va voir dans la suite)
— Méthodes de Laplace

2.1 Méthodes variationelles

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 $p(\theta|D)$. La distance choisit ici est la divergence de Kullback-Leibler :

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

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

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

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

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 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 vraissemblance des données D sachant le paramètre θ,
— $p(\theta)$ la distribution a priori de θ,
— $p(D)$ la distribution des données d'entrainement.

3 Algorithme BNN-ABC-SS

3.1 ABC (Approximate Bayesian Computation)

La méthode ABC consiste à evaluer $p(\theta|D)$ sans évaluer la vraissemblance 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

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

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

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 cotés par $\hat{y}$ on obtient notre approximation de $p(\theta|D)$

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

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

Cependant lors de l'algorithme de rejet, tirer les θ de manière générique donne trop rarement des $\hat{y}$ qui tombe 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

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

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

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.

A chaque itération, on tire à partir (conditionnées) des tirages précédents avec un ϵ_i plus fin.

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

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

3.3 Pseudo - Code

Pour la génénation des chaines de markov dans l'espace $\mathcal{N}_{\epsilon_j}(D)$, On utilise l'Algorithme 'Modified Monte Carlo' [2, 1].

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

— On genère $\theta' \sim q(\theta|\theta)$ et $y' \sim p(x|\theta')$

— On accepte (θ', x') en tant que $(\theta^{(n)}, x^{(n)})$ avec un 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 $\mathcal{N}(0, I)$

— $q(.|b)$ est celle de $\mathcal{N}(b, \sigma_0)$

Algorithme 1 : Pseudo code BNN - ABC - SS

Entrées : $N \in \mathbb{N}^*$: le nombre de tirage à chaque itération
$l_{\max} \in \mathbb{N}^*$: le nombre d'itération maximale
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 cette 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 $\theta_k^{(j-1),1} = \theta_k^{(j-1)}$ tq $\hat{y}_k^{(j-1)} \in \mathcal{N}_{\epsilon_j}(D)$
    On génère iP_0 états d'une chaine de markov de θ tq $\hat{y} \in \mathcal{N}_{\epsilon_j}(D)$:
    $\left[\theta_k^{(j-1),1}, ..., \theta_k^{(j-1),iP_0}\right]$ 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 Réalisation

On se donne un base de données à étudier avec une comparasion avec des méthodes déja existante [2, 3, 4] Ici on propose de reproduire un résultat vu dans l'article [3].

4.1 Cosinus perturbé

Les données d'entrainement $D = \{x_i, y(x_i)\}_{i=1}^{200}$, 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, un couche cachée à deux cellules et une couche de sortie ce qui fait 7 paramètres avec la fonction sigmoïde comme fonction d'activation.

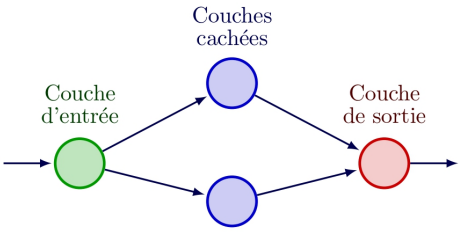


FIGURE 3 – Réseau de neurones pour la fonction cos

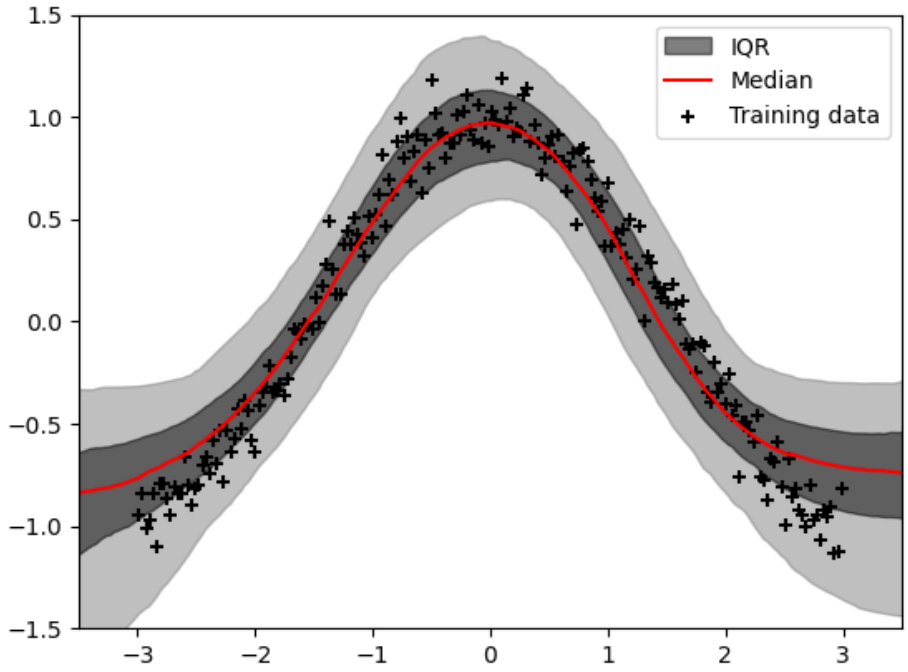
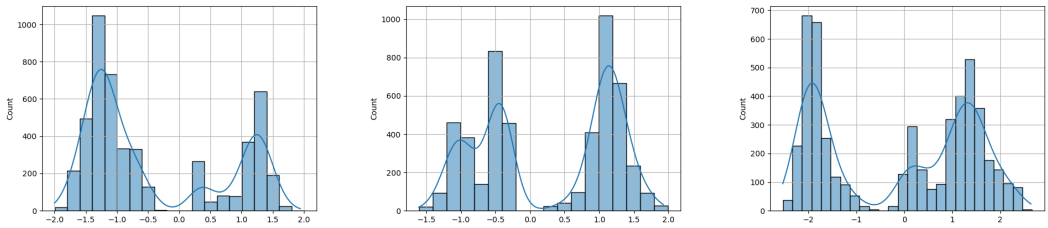


FIGURE 4 – Résultat de l'algorithme BNN-ABC-SS



(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

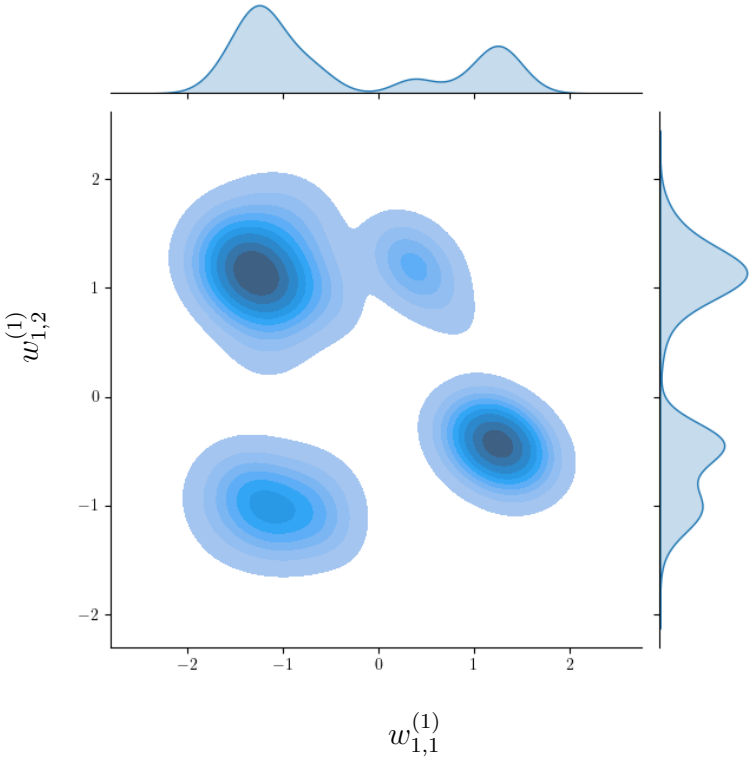


FIGURE 6 – distribution a posteriori de $w_{1,1}^{(1)}$ et $w_{1,2}^{(1)}$

4.2 Sinus perturbé

Les données d'entrainement $D = \{x_i, y(x_i)\}_{i=1}^{100}$, avec $(x_i)_{i=1}^{N}$ 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 $N : 20000$, $l_{max} : 8$, $P_0 : 0.1$ et $\epsilon : 0.1$.

Le réseau de neurones pris ici a une couche d'entrée, deux couche cachées à 15 cellules et une couche de sortie ce qui fait 286 paramètres avec la fonction ReLu comme fonction d'activation.

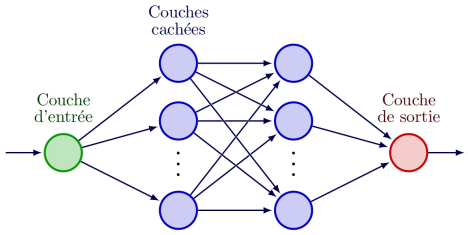


FIGURE 7 – Réseau de neurones pour la fonction sin

4.3 Maybe more

Insert im here
list hypperametters
Insert res here

5 Conclusion et perspective

On a vu ici l'implémentation d'une méthode d'approximation sur l'incertitude sur la sorite qui poura être utilisé par la suite dans ....

Références

[1] 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.

[2] 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.

[3] 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.

[4] 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.