POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Rapport de stage

LUCAS NOORMAMODE
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Aout 2022

REMERCIEMENTS

Ce stage n'aurait pas été aussi enrichissant sans les discussions et les reflexions avec mon collègue et partenaire Guillaume Desermeaux. Je tiens également à exprimer ma reconnaissance à Nicolas Piché pour ses conseils ainsi qu'à Jocya Hamelin qui a été d'un soutien sans faille à moi ainsi qu'aux autres stagiaires.

Mes remerciements vont également à mon directeur d'étude Ettore Merlo pour son appui durant mon parcours académique à Polytechnique Montréal.

RÉSUMÉ

Le contrôle et l'optimisation des processus de production est une problématique fondamentale dans l'industrie actuelle. Avec une compétition mondiale toujours plus rude et un contexte de changement climatique qui pousse les entreprises à réduire leur consommation d'énergie [4], améliorer l'utilisation des outils de production est d'autant plus intéressant. Un système de contrôle performant permet d'augmenter significativement la durée de vie des machines tout en permettant des produits finaux d'une qualité supérieure. Cela est particulièrement vrai pour les industries de pointes à forte intensité de capital comme l'aéronautique ou encore la micro-électronique.

Le principe général du contrôle machine consiste à se servir d'en ensemble de signaux provenant de différents capteurs afin d'estimer à chaque instant un état du système. Une fois cet état déterminé, des algorithmes de contrôle spécifiques peuvent ou non manipuler les entrées du système afin de le réguler et de l'amener à l'état souhaité. Ce processus est encadré par des contraintes de précision, de rapidité et de stabilité qui peuvent être spécifiées par l'utilisateur.

Les méthodes de contrôle classiques sont basés sur les théorèmes mathématiques des systèmes non linéaires [9]. Parmi les plus communes, on peut citer les contrôleurs PID, les contrôleurs par retour d'état ainsi que la commande adaptative. Aujourd'hui, avec l'émergence de l'intelligence artificielle et en particulier des méthodes reposant sur les réseaux de neurones, de nouvelles techniques de contrôle machine se sont développées avec une performance accrue, particulièrement lorsque les signaux d'états du système sont de grande dimension. On peut prendre pour exemple l'industrie micro-électronique où la production de processeurs implique des centaines de processus différents, chacun ayant plusieurs milliers de paramètres libres. La réduction de dimension, les algorithmes de classification et ceux de régression sont autant d'outils utiles pouvant être mis à profit pour déterminer l'état du système. On peut également mentionner les méthodes de contrôle par rétroaction basées sur l'apprentissage par renforcement [11] ayant pour but de prédire la meilleure action que le contrôleur devrait prendre au pas de temps suivant.

On peut distinguer deux phases dans le contrôle machine. Il faut d'abord déterminer l'état du système, puis, par la suite, déterminer quelles sont les entrées à envoyer à la machine au prochain pas de temps. Notre étude s'intéresse à résolution de la première problématique au travers du développement d'un modèle permettant de reproduire fidèlement le comportement d'une "machine X". Dans la littérature, on appelle ces modèles des *Surrogate models* ou encore des *Meta-modèles* [6]. Notre solution propose d'utiliser des méthodes d'apprentissage

supervisé (en l'occurrence des réseaux de neurones) entraînés avec des données obtenues sur le système réel. Une fois le modèle entraîné, on peut l'utiliser pour simuler des expérimentions correspondantes aux paramètres d'entrée souhaités au lieu de les réaliser dans la réalité. On considère la machine comme une boite noire. Ce cadre nous permet de ne pas faire d'hypothèses sur le fonctionnement de celle-ci. On pourrait donc envisager d'adapter la solution à d'autres problèmes similaires.
La simulation est motivée par les difficultés en termes de coûts et de contraintes d'utilisation de la machine, la rapidité d'exécution des expériences simulées ainsi que la possibilité de lancer plusieurs expériences en parallèle. La réalisation d'un simulateur pourrait également aider à la compréhension des phénomènes physiques régissant le fonctionnement d'un tel système.

Étant donné le coût d'utilisation de la machine, il est souhaitable de minimiser le temps durant lequel elle fonctionne, et donc, d'obtenir les données les plus qualitatives possibles à partir des expériences que l'on mène sur celle-ci. Dans notre cas, où nous possédons peu de données labellisées, ce besoin revient à optimiser le parcours de l'espace d'entrée, c'est à dire à déterminer itérativement quelles entrées passer à notre machine afin que la combinaison {entrées, sorties} soit la plus informative. Répondre à cette question est justement tout l'enjeu de l'*apprentissage actif*, sous-domaine de l'apprentissage machine [14].

Pour le parcours de l'espace d'entrée, nous avons décidé retenir la méthode de la "requête par votes" (*Query by Committee*). Elle consiste à entraîner en parallèle plusieurs simulateurs avec des données légèrement différentes (méthode *bootstrap*), puis à réaliser avec chacun, des prédictions. L'idée est alors de mesurer le désaccord entre les prédictions des différents simulateurs dans des zones de l'espace d'entrées. On suppose que plus le désaccord est grand dans une zone, plus celle-ci est critique pour l'apprentissage du simulateur. Notre solution se décompose en 3 variantes : une méthode dite "standard", une méthode "synthèse de requête" et une dernière "synthèse de requête avec expert de variance".
Dans la première, le désaccord est évalué sur un échantillon défini de points. Ceux pour lesquels l'écart de prédiction des membres du comités est le plus grand sont alors retenus comme prochaines entrées.
Dans la deuxième, on converge vers les points de désaccords importants par le biais d'une optimisation, ce qui permet de se passer d'un échantillon de points.
La dernière approche appelée NA-QBC (*Neural adjoint Query-by-Committy*) [12] est similaire à la précédente, mais, au lieu de converger directement vers les points de grands désaccords, on le modélise par un "expert de variance" avant d'optimiser sur ce modèle additionnel directement. Cet ajout pourrait être garant d'une convergence plus rapide en grandes dimensions.
Les 3 modèles que nous proposons donnent des résultats équivalents, voire légèrement meilleurs que l'exploration aléatoire de l'espace d'entrée. Cette performance est prometteuse et devra

être confirmée dans des dimensions supérieures à celles pour lesquelles nous avons expérimentés (>2) car c'est dans ces configurations que nos méthodes sont théoriquement les plus intéressantes. Notre solution rend par ailleurs possible l'envoi direct des entrées à la "machine X", en parallèle de la mise en oeuvre de l'apprentissage actif. Avantage non négligeable car cela permet d'éviter d'avoir à régler manuellement la machine à chaque itération. Le processus global en est grandement accéléré.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

QBC	Query by Committee
NA-QBC	Neural adjoint Query-by-Committy
ORS	Object Research Systems
ETS	École de technologie supérieure
MSE	Mean Square Error
MLP	Multi Layer Perceptron
IDE	Integrated Development Environement

LISTE DES ANNEXES

CHAPITRE 1 MISE EN CONTEXTE DU STAGE

1.1 Description de l'organisation

Object Research Systems (ORS) est une entreprise basée à Montréal spécialisée dans l'analyse et la reconstruction d'images. Membre de *Comet Group*, elle apporte également son expertise technique à d'autres entreprises du groupe Comet, entre autres dans le domaine de l'intelligence artificielle. L'entreprise est de taille moyenne avec un effectif d'une cinquantaine de personnes.

1.2 Contexte du stage

ORS est actuellement en plein phase d'expansion. Avec son expérience acquise dans son coeur de métier : le développement logiciel spécialisé dans l'analyse d'image, elle veut maintenant apporter son expertise dans d'autre domaines d'applications de l'intelligence artificielle. Un des marchés qu'elle vise est celui du contrôle de processus. Cette discipline a pour but d'optimiser le fonctionnement de systèmes de production en termes d'efficacité, de vitesse et de sécurité. Dans les industries, le contrôle de processus est utilisé pour améliorer les performances des chaînes de production et ainsi faire des économies de ressources.

Une des sociétés soeurs d'ORS (que nous nommerons entreprise X) est spécialisé dans le domaine de l'électronique de pointe, en particulier dans la mise au point d'appareils de production complexes. De nombreuses applications existent, en particulier dans l'industrie métallurgique et dans la micro-électronique notamment pour réaliser des traitements de surfaces de grande précision.

L'entreprise X a sollicité l'aide d'ORS pour optimiser un procédé industriel particulier utilisé dans la fabrication de circuits intégrés, notamment lors d'étapes de gravures *(Etching)* et de dépots *(Deposition)* [5].

Pour ORS, ce projet est capital car il s'agit d'un des premiers partenariats au sein du groupe Comet et hors de leur domaine d'expertise direct. Cependant, il est encore dans sa phase initiale, seules les grandes lignes et les principaux défis ont été identifiés par l'entreprise avant

le début du stage. De plus, le calendrier chez l'entreprise X ne prévoit le commencement du projet qu'a partir de septembre 2022, Plusieurs éléments nécessaires au début effectif de la phase de développement sont donc absents. Il manque ainsi notamment une base de données {entrées, sorties} de la machine X.

Dans ce contexte, la direction R&D d'ORS a décidé de confier à deux stagiaires (un élève de l'École de technologie supérieure (ETS) et moi-même) ainsi qu'à un titulaire le soin de mener les étapes préliminaires du projet, c'est à dire identifier les principales pierres d'achoppements et résoudre ce qui peut l'être sans les données cruciales de l'entreprise X.

1.3 Mandat du stage et responsabilités de la personne stagiaire

Le mandat du stage recouvrait deux problématiques distinctes. D'abord, je devais avec mon équipe anticiper, décrire et faire l'état de l'art des problèmes que pourrait rencontrer ORS dans le développement du simulateur de la machine X. Puis, dans un second temps, le but était de résoudre un ou plusieurs de ces problèmes dans la limite de ce qu'il est possible de résoudre sans la collaboration active de l'entreprise X.

Durant la première phase d'investigation, l'équipe travaillait de façon très proche et il n'est pas possible de distinguer de responsabilité distincte. Durant la seconde phase, j'avais pour but de résoudre la problématique d'apprentissage actif en elle-même (qui sera décrit dans le chapitre 3 du présent rapport) quand mon collègue stagiaire s'est lui concentré sur l'implémentation du contrôle à distance de la machine X et à la mise en parallèle des calculs nécessaire à l'apprentissage actif.

L'environnement de travail était convivial avec une équipe experte. Une grande liberté et confiance nous a été accordée ce qui a permis l'exploration nécessaire au développement de notre solution.

1.4 Déroulement du stage

Malgré le contexte pandémique, la majorité du stage a pu se dérouler en présentiel, dans les locaux de l'entreprise. La phase d'investigation a duré environ un mois, de mi mai à mi juin, quand la phase de résolution du problème de l'apprentissage actif m'a occupé le reste du stage.

Il est a noter que la personne titulaire sensé faire partie de notre équipe n'a pas pu terminer le projet suite à un évènement hors de notre contrôle. Nous étions donc seul (chacun sur sa problématique) pour la phase de résolution des solutions.

Tableau 1.1 Détails du déroulement du stage

Durée	Activité réalisées	Équipe
1 semaine	Intégration dans l'entreprise et prise en main des outils	Moi-même
2 semaines	Prise en en main du problème	Moi-même
1 semaine	Revue de littérature sur la machine X	Complète
1 semaine	Revue de littérature sur l'apprentissage actif	complète
1 semaine	Mise en commun, identification des tâches et répartition de celles-ci	Complète
7 semaines	Réalisation de la solution d'apprentissage actif	Moi-même
1 semaine	Mise en commun des travaux	Moi-même et stagiaire ETS

CHAPITRE 2 RETOUR D'EXPÉRIENCE

2.1 Retour sur les objectifs concernant les compétences professionnelles et techniques

2.1.1 Consolider ses connaissances et ses habiletés par la mise en application de notions, de méthodes et d'outils vus pendant sa formation aux cycles supérieurs et applicables à une problématique d'intérêt pour l'industrie

J'ai pu consolider mes connaissances en mathématiques pure, particulièrement dans les domaines des probabilités et statistiques et d'algèbre linéaire. Au niveau informatique, j'ai pu mettre en oeuvre des connaissances en apprentissage machine et en recherche opérationnelle. D'un point de vue logiciel, j'ai perfectionné mes compétences en codage Python, en architecture logicielle et en programmation orienté objet.

2.1.2 Acquérir des connaissances pratiques pertinentes à sa discipline en utilisant les approches, méthodes et systèmes de l'organisation d'accueil

J'ai pu observer comment se déroulait le processus de développement et de correction de bugs d'un logiciel dans sa phase de développement avancée (>10 ans). J'ai eu l'occasion de voir comment s'effectue le déploiement pour production d'un tel logiciel. J'ai pu utiliser l'ensemble des outils et systèmes informatiques utilisé quotidiennement au sein de l'organisation notamment au niveau des Integrated Development Environement (IDE), de la gestion des fichiers, de l'organisation des tâches.

2.1.3 Développer son sens critique à l'égard des approches, méthodes et systèmes utilisés dans le milieu de travail

J'ai pu discuter avec des membres de longue date de l'organisation qui m'ont expliqué les erreurs que l'entreprise avait pu faire dans le passé selon eux. J'ai pu constater différentes tentatives d'amélioration continue dans l'organisation. J'ai également pu être force de proposition en indiquant à mon superviseur certaines méthodes encore utilisées par l'entreprise mais plus d'actualités dans le domaine.

2.1.4 Découvrir un domaine d'application, acquérir de l'expérience de travail, développer des contacts professionnels, et augmenter ainsi son employabilité

J'ai pu découvrir le domaine du développement logiciel, en particulier dans le secteur du traitement et de l'analyse d'images. Mon stage à également débouché sur une offre d'emploi.

2.2 Retour sur les objectifs concernant les habilités personnelles et relationnelles

2.2.1 Développer ou améliorer ses habilités de communication écrite et orale

J'ai pu échanger dans un environnement bilingue Français/Anglais à la fois à l'oral et à l'écrit. J'ai pu améliorer ma capacité de précision et de synthèse en devant notamment expliquer mon projet ainsi que les différents problèmes que je pouvais rencontrer à des membres de l'équipe non impliqués dans le projet.

2.2.2 Effectuer une autoévalutation de ses habilités personnelles et relationnelles résultant du stage effectué

J'ai pu développer mes habilités personnelles en échangeant avec de personnes de tous âge et avec des parcours très différents. J'ai du gérer avec d'autres stagiaires un conflit avec un collègue. J'ai pu échanger avec des partenaires de l'entreprise et ainsi découvrir les difficultés de communications qui peuvent exister même au sein d'une même organisation. J'ai su développer des amitiés et de mon point de vue des relations de travail très positives avec les personnes avec lesquelles je collaborais. J'évalue donc mes habilités personnelles et relationnelles à bonnes. Mes axes d'améliorations se situent au niveau de la reconnaissance et de la gestion de mes émotions.

2.3 Évaluation des apprentissages du stage en lien avec le programme d'études

Mon stage se déroulait dans mon domaine d'étude. J'ai pu mettre en oeuvre des techniques d'apprentissage machine en lien direct avec ce que j'ai appris. J'ai également mobilisé une grande partie de mes connaissances en mathématiques, spécifique à mon programme d'étude. J'ai enfin pu utiliser le langage de programmation *python* ainsi que l'environnement *tensorflow* qui sont parmi les outils les plus répandus dans le domaine actuellement.

CHAPITRE 3 PROBLEMATISATION D'UN VOLET DU STAGE

Dans ce chapitre, nous allons présenter le travail effectué lors de la seconde partie du stage, c'est-à-dire la résolution d'un problème d'apprentissage actif ayant pour but la constitution optimale d'un base de données pour l'apprentissage d'un simulateur de la machine X. Nous commencerons par mettre en contexte le problème au sein du projet dans lequel il s'inscrit, spécifier les objectifs que nous souhaitions atteindre et faire l'état des lieux de la littérature pertinente. Puis, nous présenterons les différentes approches que nous avons retenu, décrirons nos résultats, avant de les discuter et d'indiquer quel est le travail restant avant de pouvoir pleinement valider notre solution.

3.1 Mise en contexte et problématique étudiée

3.1.1 Mise en contexte

Cette solution s'inscrit dans le projet décrit plus tôt avec le partenaire industriel d'ORS : Entreprise X. Pour rappel, le but du projet est la réalisation d'un simulateur de machine X c'est à dire un *Surrogate model* [6]. La création de cette simulation est motivée par plusieurs raisons. D'abord, la machine est très contraignante à utiliser, entre autres, à cause de son coût élevé de fonctionnement et des normes de sécurité à respecter. Pouvoir mener des expérimentations sur le simulateur plutôt que sur le système réel pourrait donc être très intéressant pour l'entreprise X. De plus, il serait possible de lancer plusieurs expériences en parallèle, chacune d'entre elles bien plus rapide par rapport à ce qui se fait actuellement. Enfin, la physique sous-jacente des machines X n'est pas encore bien connue, la réalisation d'un simulateur pourrait donc aider à la compréhension des phénomènes s'y déroulant.

Nous avons décidé pour la réalisation du simulateur d'utiliser l'apprentissage supervisé [8] en considérant la machine comme une boite noire. Les seules informations auxquelles nous avons accès sont donc les entrées et les sorties de la machine.

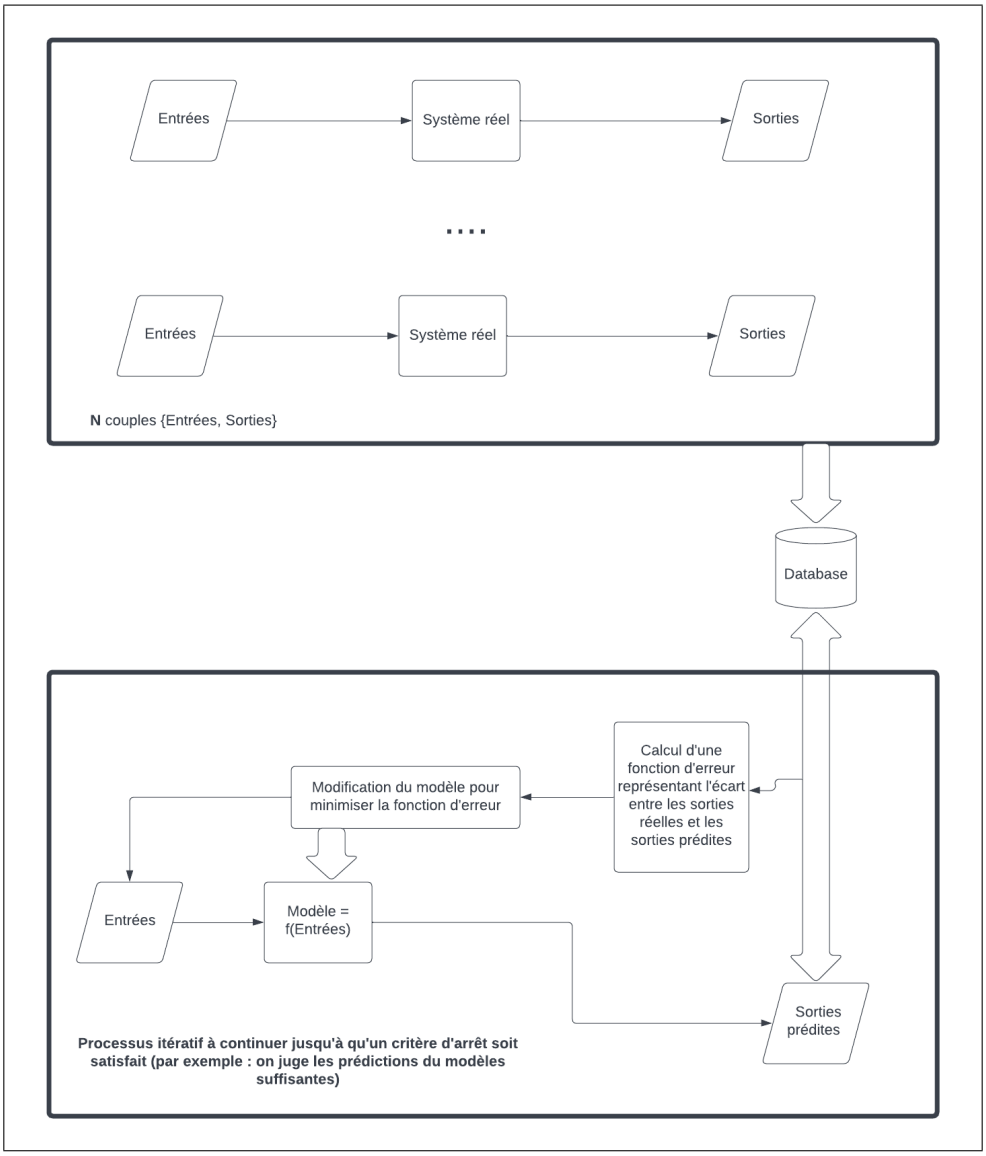

FIGURE 3.1 Schéma de principe du processus d'apprentissage supervisé

Comme on peut le voir dans la figure 3.1, une base de données {Entrées, Sorties} est un ingrédient indispensable de tout processus d'apprentissage supervisé. Dans notre cas, cette base de données n'existe pas encore, il nous faut donc la créer en envoyant des entrées à la machine réelle (cela correspond à un réglage de la machine) qui nous renverra des sorties (cela correspond à son état suite à son réglage). Quand on envoie des entrées à la machine et qu'on observe sa sortie, on dit que l'on évalue un point.

La machine possède 14 paramètres de réglages qui sont semblables à ceux que l'on peut trouver par exemple sur la figure 3.2 qui représente une machine d'une complexité similaire

mais qui possède, elle, 27 degrés de liberté.

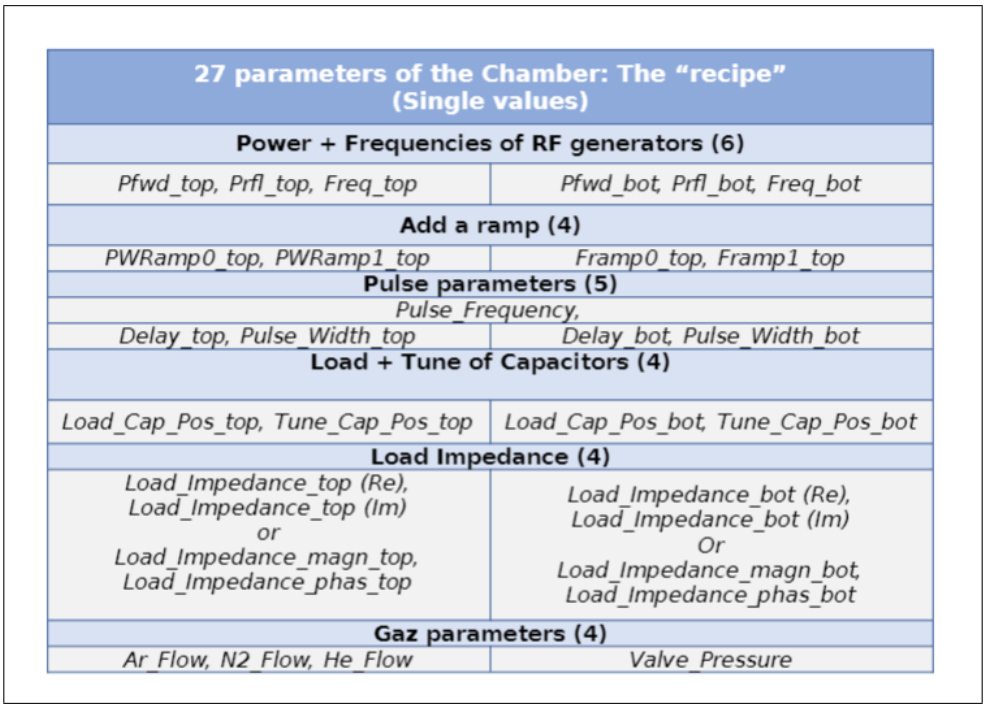

FIGURE 3.2 Exemple de réglage pour une machine de complexité similaire à la machine X

En général, quand on doit constituer une base de données, on peut parcourir l'espace en constituant une grille dans l'espace d'entrée comme dans les figures 3.3 et 3.4 qui montrent un exemple de cette méthode respectivement en 2 et 3 dimensions.

Malheureusement, cette approche n'est pas réalisable en 14 dimensions en raison du nombre exponentiels de points à évaluer en fonction du nombre de dimensions (voir tableau 3.1).

TABLEAU 3.1 Nombre de points à évaluer en fonction du nombre de dimension pour une densité de 100 points par dimensions

Dimensions	Nombre de points à évaluer
1	100
2	10 000
3	1 000 000
5	10 000 000 000
14	100^{14}

Une autre approche classique est de tout simplement prendre des points aléatoires dans l'espace d'entrée.
Cela n'est ici pas non plus souhaitable en raison de la nature de l'objet que l'on souhaite

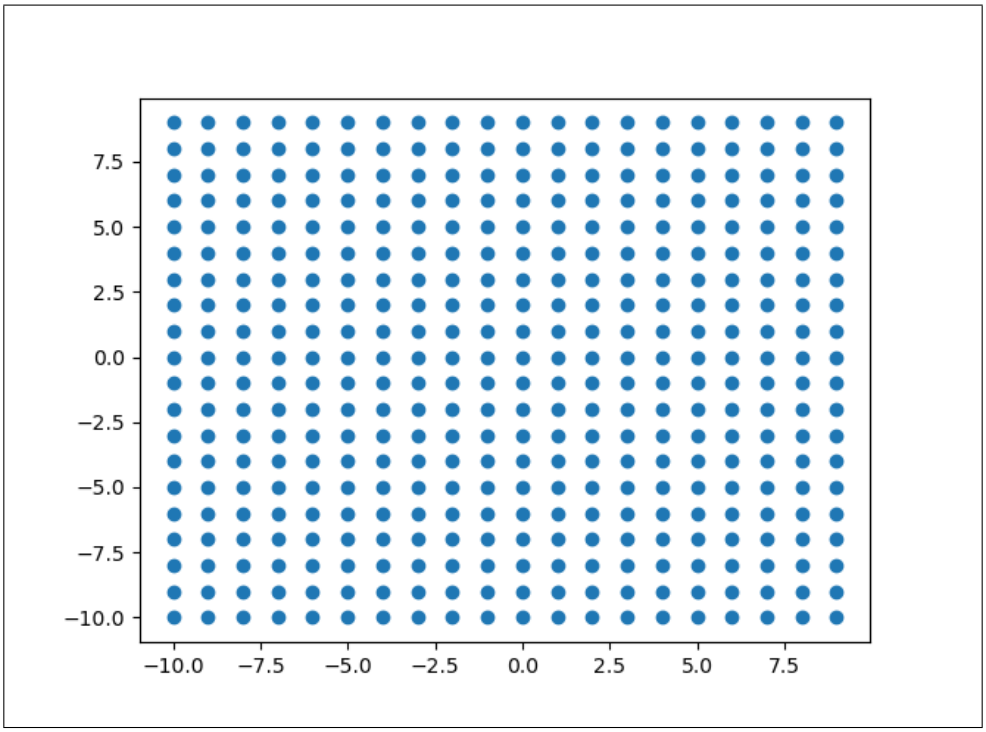

FIGURE 3.3 Exemple de grille en 2 dimensions

apprendre. En effet, on sait que la réponse de la machine X par rapport aux entrées est très non linéaire. Ainsi, il n'est pas déraisonnable de faire l'hypothèse que certaines zones de l'espace d'entrée sont plus "denses" en information que d'autres, c'est à dire des zones telles que :

$$f(\mathbf{x}) \text{ est très différent de } f(\mathbf{x} + \epsilon)$$

avec $\mathbf{x}$ le vecteur contenant une entrée et ϵ un vecteur de petites variations.

Ces zones sont critiques pour l'apprentissage car elles peuvent concentrer l'essentiel de l'écart entre les prédictions et le système réel (La majorité des modèles d'apprentissage supervisé apprennent facilement les relations linéaires). Pour l'optimiser, il faut donc avoir plus de données dans ces zones par rapport à celles plus linéaires. Tirer aléatoirement des points de l'espace d'entrée ne le permet pas.

Un autre désavantage des méthodes que nous venons de décrire est qu'elles ne sont pas "actives". Elles ne ne considèrent pas les points déjà présents dans la base de données, ni les performances de prédiction que le modèle peut atteindre avec ces points déjà évalués.

Le constat que nous venons de faire motive l'utilisation de méthodes plus avancés de parcours de l'espace d'entrée permettant de combler les écueils des méthodes classiques.

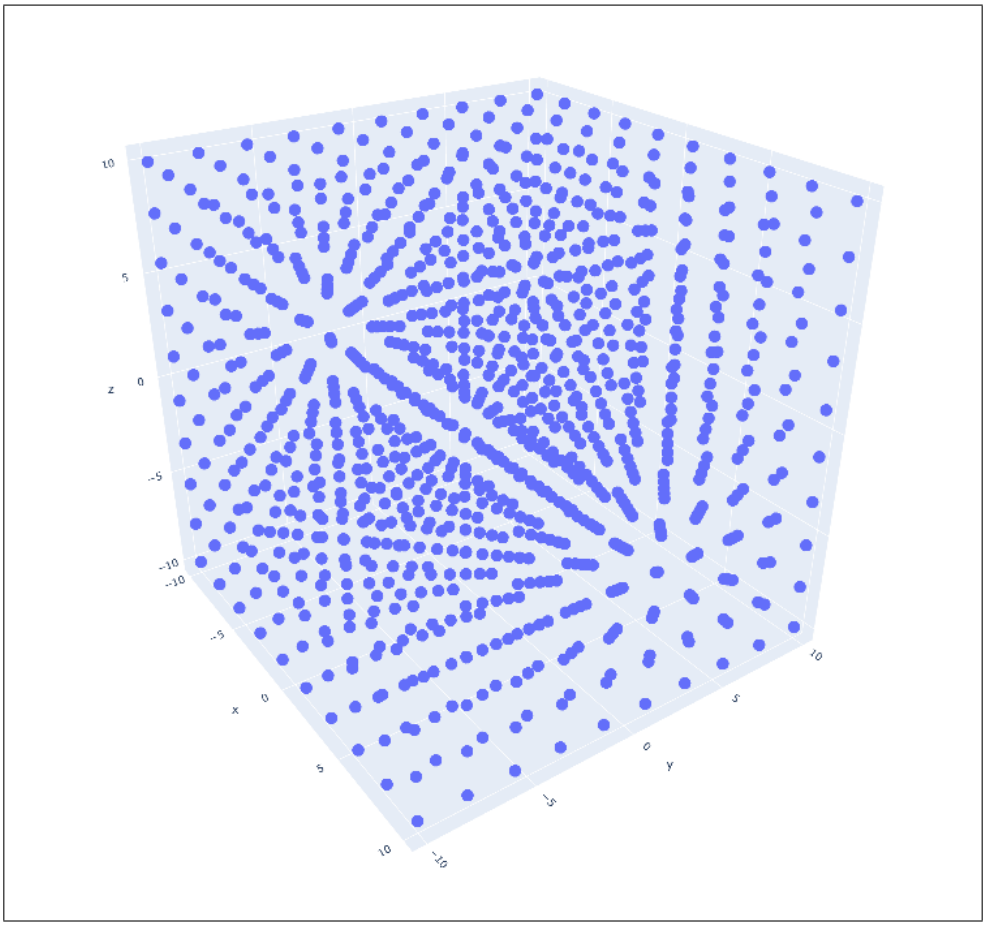

FIGURE 3.4 Exemple de grille en 3 dimensions

3.1.2 Problématique

Maintenant que nous avons mis en contexte le problème que nous souhaitons résoudre il est possible d'expliciter notre problématique.

Soient $d \in \mathbb{N}$ la dimension des entrées de la machine X, $p \in \mathbb{N}$ la dimension des sorties de la machine X
Soient $\forall i \in [\![1; d]\!]$, E_i l'ensemble des entrées possibles dans la dimension i.
Soient $\forall i \in [\![1; p]\!]$, S_i l'ensemble des sorties possibles dans la dimension i.
Soit $X = E_1 \times E_2 \times ... \times E_d$, l'espace d'entrée.
Soit $Y = S_1 \times S_2 \times ... \times S_p$, l'espace de sortie.
Soit $\mathbf{x} = (x_1, x_2, ..., x_d) \in X$ le vecteur des entrées de la machine.
Soit $\mathbf{y} = (y_1, y_2, ..., x_p) \in Y$ le vecteur des sorties de la machine.

Soit f la fonction de X dans Y représentant la machine X (appelée *oracle*) :

$$\begin{array}{rccc} f: & X & \longrightarrow & Y \\ & \mathbf{x} & \longmapsto & \mathbf{y} \end{array}$$

Soit $\hat{f}$ l'estimation de la fonction f par un modèle quelconque :

$$\begin{array}{rccc} \hat{f}: & X & \longrightarrow & Y \\ & \mathbf{x} & \longmapsto & \hat{\mathbf{y}} \end{array}$$

avec $\hat{\mathbf{y}}$ l'estimation de $\mathbf{y}$ par $\hat{f_\theta}$ Soit l une fonction de perte :

$$\begin{array}{rccc} l: & Y^2 & \longrightarrow & \mathbb{R} \\ & (\hat{\mathbf{y}}, \mathbf{y}) & \longmapsto & l(\hat{\mathbf{y}}, \mathbf{y}) \end{array}$$

mesurant l'écart entre la valeur réelle de la sortie et la valeur prédite par le modèle.
Soit D_n une base de données :

$$D_n = \{(\mathbf{x_1}, \mathbf{y_1}), (\mathbf{x_2}, \mathbf{y_2}), ..., (\mathbf{x_n}, \mathbf{y_n})\}$$

Elle correspond à n réalisation de couples de variables aléatoires $D_n = \{(\mathbf{X_1}, \mathbf{Y_1}), (\mathbf{X_2}, \mathbf{Y_2}), ..., (\mathbf{X_n}, \mathbf{Y_n})\}$ que nous supposons indépendantes et identiquement distribués.

Dans notre problème, nous souhaitons agrandir notre base de données. Appelons ce nouvel ensemble D_{new} :

$$D_{new} = D_n \cup D_m$$

avec $m \in \mathbb{N}$ le nombre de nouveaux éléments à ajouter et D_m, l'ensemble de ces éléments.

Cette base de données étendue va mener à une nouvelle estimation de la fonction f : $\hat{f_{new}}$.

Notre but est de déterminer l'ensemble D_m tel que la nouvelle estimation $\hat{f_{new}}$ soit *optimale*, c'est à dire trouver l'ensemble D_m^* tel que :

$$D_m^* = \underset{D_m}{argmin} \sum_{(\mathbf{x},\mathbf{y}) \in D_{new}} l(\hat{f}_{new}(\mathbf{x}), \mathbf{y})$$

D_m^* est composé de m couples d'entrées $\mathbf{x_i}$ et de sorties $\mathbf{y_i}$ avec $i \in [\![1, m]\!]$. On appelle l'ensemble des $\mathbf{x_i}$ la requête (*query*). On peut considérer la requête comme étant un ou plusieurs points dans l'espace d'entrée X. L'action d'évaluer les $\mathbf{x_i}$ par f afin d'obtenir les

$\mathbf{y_i}$ est la *labelisation*. Les processus pour déterminer iterativement les requêtes forment ce que l'on appelle les méthodes d'apprentissage actif. Approcher l'ensemble D_m^* revient donc à approcher les entrées qui le composent car on peut directement labéliser les sorties à partir de celles-ci.

En plus de simplement approcher l'ensemble D_m^*, nous devons ajouter une contrainte à la méthode que nous voulons développer. En effet, la requête optimale dépend de la nature et/ou des paramètres de $\hat{f}$. Par exemple, pour un modèle de régression polynomial, du degré du polynôme ou encore pour un modèle neuronal, de la méthode d'apprentissage, du nombre de couches etc...Dans notre cas, comme vu dans la présentation du contexte du stage, il nous manque des informations en provenance de l'entreprise X qui auraient pu nous permettre de faire des hypothèses sur la forme de $\hat{f}$. Sans cet apport, il serait préférable que la façon dont nous approchons D_m^* soit invariante par à $\hat{f}$.

3.2 Objectifs poursuivis

Les objectifs poursuivis sont les suivants.

1. Déterminer une méthode permettant d'approcher pour une base de données comportant $n \in \mathbb{N}$ élements : D_n, la requête optimale à $m \in \mathbb{N}$ éléments.
   - Cette méthode devra au minimum avoir une performance similaire par rapport à la formulation de requêtes aléatoires, ce qui correspond à la méthode la plus basique.
   - Il serait préférable que cette méthode soit la plus invariante par rapport à la façon dont nous construisons $\hat{f}$.
2. L'implémentation en code de cette méthode devra être capable de communiquer avec une machine réelle.
3. La méthode devra pouvoir générer des requêtes dans l'ensemble de l'espace.

3.3 Revue documentaire

3.3.1 Les scénarios de l'apprentissage actif

Dans le domaine de l'apprentissage en ligne, il existe plusieurs scénarios pour la génération des requêtes.

— *Synthèse des requêtes (Membership Query Synthesis)* : Les requêtes sont fabriquées à partir de règles prédéfinies pour qu'elles appartiennent à la distribution de l'espace d'entrée du modèle. Ainsi, la génération des requêtes est possible dans l'ensemble continu de l'espace d'entrée. C'est l'un des premiers scénarios à être étudié.

Par exemple, il a été utilisé pour apprendre les coordonnées absolues d'un bras robot à partir des angles de ses articulations [3]. Un désavantage de cette approche est que, pour certains problèmes, les requêtes sélectionnées peuvent ne pas exister dans la réalité ce qui peut les rendre impossible à labéliser.

— *Échantillonage par sélection de flux (Stream-based Selecting Sampling)* : Cette méthode repose sur l'hypothèse qu'obtenir un élément de l'espace d'entrée du modèle est très peu coûteux. Les potentielles requêtes vont ainsi tour à tour être examinées par un algorithme qui va décider si elles sont intéressantes à garder pour labélisation.

— *Échantillonage par ensemble (Pool-based Sampling)* : Ce dernier scénario nécessite de posséder ou de créer un ensemble fini de requêtes potentielles parmi lesquelles on va sélectionner les plus intéressantes à labéliser. Pour se faire, un critère va être utilisé pour chaque élément. Cette approche est très populaire car elle est facile à mettre en place et permet de faire un choix sur des critères de comparaison parmi plusieurs échantillons. Cela permet donc d'obtenir les "meilleures" requêtes plutôt que des "bonnes" requêtes qui résultent de l'échantillonnage par sélection de flux. Ce scénario a par exemple été étudié pour faire de l'apprentissage actif pour des problèmes de régression avec du *Greedy Sampling* [16].

Dans la suite de cette section, nous parlerons des méthodes les plus courantes de l'apprentissage actif pour répondre à des problèmes de classification ou de régression. Puis nous détaillerons plus précisément le fonctionnement des méthodes de *requêtes par votes* (ou *Querry-By-Committe (QBC)*) qui seront étudiés dans la suite de cette étude.

3.3.2 Echantillonnage par incertitude (ou *Uncertainty Sampling*)

L'échantillonnage par incertitude est la plus simple et la plus populaire des méthodes d'apprentissage actif. Elle repose sur le principe que si le modèle est très incertain du label à attribuer à une requête potentielle, alors celle-ci est informative et doit être sélectionnée pour être labélisée [13]. Cette approche fonctionne particulièrement bien pour les modèles probabilistes. Par exemple, lors d'un problème de classification bi-classe, on peut considérer que si les probabilités liées à chacune sont proches de 0.5, l'incertitude est forte. On peut généraliser le problème en considérant plusieurs métriques permettant de décider quels échantillons sélectionner :

— *Least confident* :

$$x^* = \underset{x}{argmax}\ 1 - P_\theta(\hat{y}|x)$$

Avec $\hat{y} = \underset{y}{argmax}\ P_\theta(y|x)$, la classe avec la plus grande probabilité d'appartenance. Cette mesure permet d'obtenir l'élément parmi un ensemble avec la plus petite probabilité d'appartenir à sa classe de prédiction. De ce fait, seule la probabilité liée à la classe de prédiction est considérée.

— *Margin sampling* :

$$x^* = \underset{x}{argmin}\ P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x)$$

Avec $\hat{y}_1$ et $\hat{y}_2$ la première et seconde classe la plus probable d'après les résultats du modèle. Cette métrique permet de mieux prendre en compte les probabilités des autres classes car on considère la marge entre les probabilités des 2 classes les plus probables.

— *Entropy sampling* :

$$x^* = \underset{x}{argmax}\ -\sum_i P_\theta(\hat{y}_i|x) \times \log P_\theta(\hat{y}_i|x)$$

Cette stratégie utilise l'*entropie de Shannon* comme mesure de l'incertitude liée à une potentielle requête x. L'entropie est proportionnelle à la moyenne de la quantité d'information nécessaire pour encoder la distribution des probabilités. Cette approche est la plus populaire dans la littérature car elle permet de prendre en compte toutes les probabilités liées aux classes potentielles du problème.

Les méthodes d'échantillonnage par incertitude ne sont généralement applicables qu'aux problèmes de classification. Mais, dans certains cas particuliers, il est possible de les appliquer à des problèmes de régression (comme le Design of Experiment). Par exemple, quand des processus gaussiens sont utilisés comme modèles, le déterminant peut être utilisé comme un critère d'incertitude prédite et ainsi de gain potentiel d'information mutuelle entre les différentes variables [17].

Le principal désavantage de cette méthode réside dans son incapacité à corriger les mauvaises prédictions. En effet, si le modèle prédit avec une haute probabilité un résultat faux, alors il ne va pas chercher à faire de nouvelles requêtes dans ces zones-là. De cette façon, il va devenir confiant dans des zones mal prédites.

3.3.3 Réduction de l'erreur prévue (ou *Expected Error Reduction*)

Cette approche d'apprentissage actif est basée sur la mesure de combien l'erreur de généralisation du modèle est susceptible de baisser[13]. L'idée est d'estimer l'erreur attendu du modèle après avoir ajouté le nouvel échantillon labélisé. Pour se faire, on doit réentraîner le modèle pour chaque échantillon possible à ajouter. Par exemple, dans un contexte de classification

on va choisir la requête par :

$$x^*_{0/1} = \underset{x}{argmin} \sum_i P_\theta(y_i|x)(\sum_{u=1}^{U} 1 - P_{\theta^{+\langle x+y_i\rangle}}(\hat{y}|x^{(u)})$$

$\theta^{+\langle x+y_i\rangle}$ représente le nouveau modèle après qu'il ait été réentrainé avec le couple $\langle x + y_i\rangle$ ajouté à la base données. U est l'ensemble des requêtes potentielles.

L'avantage de cette méthode est qu'elle est quasi-optimale étant donné qu'elle s'intéresse directement à réduire l'erreur du modèle. Cependant, elle est extrêmement lourde en calculs.

3.3.4 Changement de modèle prévu (ou *Expected Model Change*)

La méthode par changement de modèle prévu s'intéresse aux points pour lesquels un modèle est le plus susceptible de modifier ses paramètres [13]. Ainsi, elle va considérer un point intéressant si l'entraînement du modèle avec celui-ci provoque un changement important des paramètres par rapport à ce qu'ils seraient en retirant ce point de la base de données. Par exemple, dans le cas d'un réseau de neurones, la quantification de ce changement peut correspondre à la somme des gradients calculés sur l'ensemble des poids du modèle. L'hypothèse sous-jacente à ce principe est, que plus un modèle va changer pour s'adapter à un nouveau point, plus celui-ci était informatif. Cela veut aussi dire que son ancienne prévision était mauvaise en ces points-là. Cette méthode est généralement efficace même si elle est très coûteuse en calculs car, comme pour la réduction de l'erreur prévue, il faut réentrainer le modèle pour chaque nouveau point potentiel.

3.3.5 Requête par votes (ou *Query by committee (QBC)*)

La méthode de requêtes par votes repose sur l'utilisation d'un comité de modèles $\mathcal{C} = (\hat{f}_1, ..., \hat{f}_n), n \in \mathbb{N}$ [15][13]. Chaque membre du comité est entraîné sur un ensemble de données labélisé. Le principe est que chaque modèle va voter sur la valeur de sortie (le label) correspondant à une potentielle requête x. Le résultat du comité est considéré comme étant la moyenne des votes y_i. On va ensuite évaluer le désaccord des modèles en calculant la variance des votes selon la formule suivante :

$$\mathcal{C} : \left\{ \begin{array}{l} \hat{f}_1(x) \to \hat{y}_1 \\ \hat{f}_2(x) \to \hat{y}_2 \\ ... \\ \hat{f}_n(x) \to \hat{y}_n \end{array} \right\} \Longrightarrow \left\{ \begin{array}{l} \mathcal{C}(x) = \frac{1}{n}\sum_{i=1}^{n} \hat{y}_i \\ Var(x) = \frac{\sum_{i=1}^{n}(\hat{y}_i - \mu)^2}{n} \end{array} \right.$$

Plus le comité est en désaccord sur le label à attribuer à x, c'est à dire plus la variance est élevé, plus cette requête est considéré comme informative. On va donc sélectionner les requêtes avec la variance la plus élevé parmi un ensemble fini de points. Le principal désavantage de cette méthode et qu'elle peut mal fonctionner si le biais du modèle est trop élevé. En effet, si le modèle est incapable de représenter correctement le véritable lien entre les variables d'entrée et de sortie, le QBC va faire des requêtes dans les mêmes zones continuellement sans que l'erreur générale ne se réduise.

Plusieurs variantes du QBC existent avec notamment le *Query-by-boosting* et *Query-by-bagging* [2] qui utilisent les méthodes de *bagging* et de *boosting* pour l'initialisation du comité. Ces méthodes ensemblistes permettent de réduire l'erreur liée aux fluctuations des données labélisés liée à la variance. En apprentissage machine, on appelle ce type d'erreur "variance" à l'opposé du "biais" qui provient directement de type de modèle utilisé qui ne peut représenter correctement les données. Par ailleurs, on peut observer ces 2 erreurs dans la décomposition de l'erreur moyenne au carré avec ici un exemple sur une fonction d'erreur quadratique.

$$E[(\hat{y} - y)^2|x] = \underbrace{E[(y - E[y|x])^2]}_{\text{bruit}} + \underbrace{(E[\hat{y}] - E[y|x])^2}_{\text{biais}} + \underbrace{E[(\hat{y} - E[\hat{y}])^2]}_{\text{variance}}$$

Ces méthodes créent plusieurs sous bases de données à partir de l'ensemble de données initial en faisant des tirages aléatoires avec remise. Elles sont ensuite utilisées pour entraîner les modèles du comité dont les résultats seront moyennés pour faire des prédictions.
Dans le cas du Bagging, les différents modèles vont faire leurs prédictions de manière parfaitement indépendante et parallèle. A l'opposé, avec la méthode par Boosting, les résultats des modèles sont incrémentalement utilisés afin de redistribuer des poids sur les données mal classifiées. Cela permet aux modèles d'avoir un meilleur apprentissage et donc une erreur généralement plus faible qu'avec un bagging mais en contrepartie, cette méthode peut être plus sensible au surapprentissage (*overfitting*) [8].

Généralement, le QBC est utilisé dans un scénario de type *Échantillonage par ensemble(Pool-based Sampling)*. On évalue donc la variance de chaque élément d'un ensemble de taille n avant de sélectionner les m avec la variance plus importante.

$$x^* = \underset{x}{argmax}\ Var(x)$$

avec x^* la requête optimale dans le cas où l'on souhaite ne sélectionner qu'un seul élément ($m = 1$).

Afin que la méthode puisse générer des requêtes dans l'ensemble de l'espace d'entrée, l'en-

semble des requêtes potentielles est souvent construit on faisant des tirages aléatoires dans celui-ci. Le nombre N de tirage est alors un paramètre important. En effet, si celui-ci est trop petit par rapport au volume d'espace à explorer, alors potentiellement une zone intéressante de l'espace peut être oubliée et donc les requêtes peuvent ne pas être assez informative. A l'inverse, si N est trop petit, alors les m points avec la variance la plus élevée peuvent se retrouver très proches dans l'espace et on se retrouve donc avec des requêtes similaires dont l'informativité n'est pas optimale. Ce dilemme est illustré par les pastilles (a) et (b) de la figure 3.5.

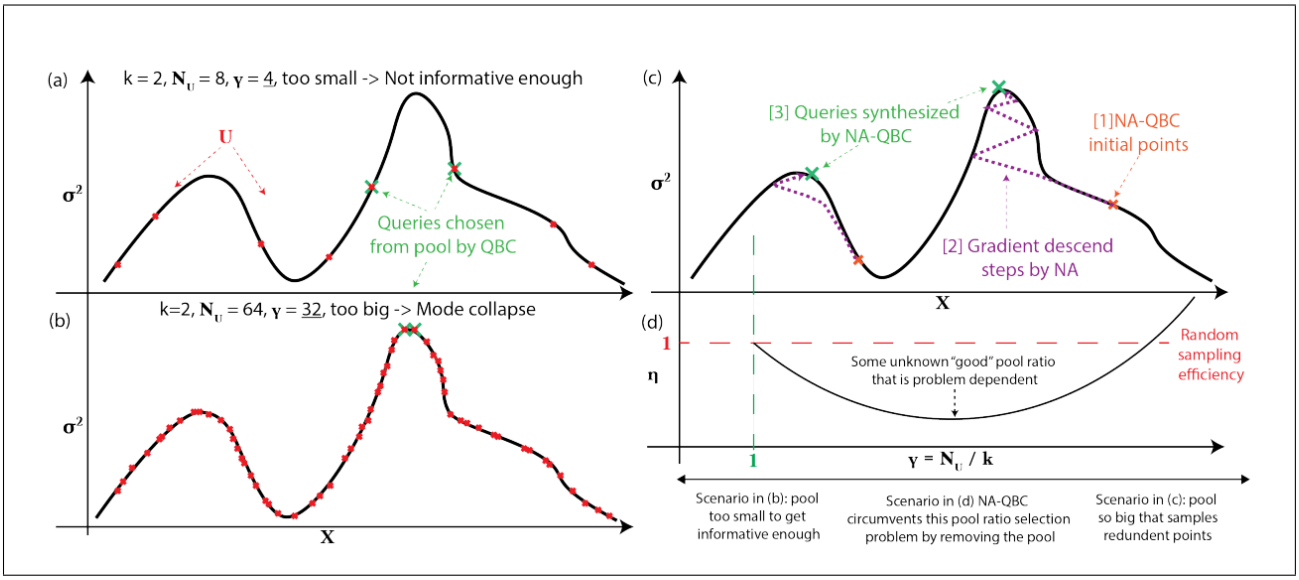

FIGURE 3.5 Figure montrant la différence entre le *QBC* avec et sans *Query Syntesis* en 1D. N_u est la taille de l'ensemble sur lequel la variance est évaluée et k le nombre de requête effectuées. (a, b) sont le résultats du QBC standard avec différents N_u. (c) présente le fonctionnement du *Query Syntesis* par optimisation. (d) Présente le dilemme du choix des paramètres k et N_u. (Tirée de [12])

3.4 Approches

3.4.1 Choix de la méthode *Query by committee* (QBC)

Suite à la revue des techniques du domaine, nous avons décidé de nous concentrer sur l'implantation des méthodes de QBC, plus particulièrement sur 3 variantes de celles-ci. Ce choix est d'abord motivé par la nature du système que nous tentons de représenter. En effet, nous sommes face à un problème de régression en grande dimension (14) et les phénomènes régissant la machine X sont complexes. Il est donc raisonnable de faire l'hypothèse que notre modèle devra être très flexible et donc gourmand en temps d'entraînement. Sachant cela,

nous avons exclus les méthodes de changement de modèle prévu et de réduction de l'erreur prévue qui nécessitent de réentrainer très souvent le modèle.
Par ailleurs, même si il est possible d'adapter l'échantillonnage par incertitude aux problèmes de régression [17], la majorité de la littérature sur le sujet se concentre sur la classification. De plus, tous les types de modèles ne sont alors utilisable et nous voulons pas faire d'hypothèses quand à leur nature.
Les méthodes QBC possèdent l'avantage d'être simples à mettre en place, de pouvoir être utilisées avec tous types de modèles et d'en plus fournir naturellement une estimation de l'erreur de prédiction directement par la mesure de la variance du comité.

3.4.2 Définition des différentes variantes du QBC

3.4.2.1 QBC standard (QBC1)

Dans cette première variante, on se place dans un scénario de *Pool-based Sampling* car on va construire un ensemble parmi lequel on va choisir les meilleures requêtes. Les requêtes ne pourront donc se faire qu'au sein de cet ensemble. Pour sa construction, on va simplement tirer un certain nombre de points dans l'espace d'entrée de façon uniforme.

3.4.2.2 QBC avec synthèse de requêtes (QBC2)

Dans ce cas-ci, la méthode est de type *Query Synthesis* car les requêtes crées ne vont pas appartenir à un ensemble de points prédéfini. L'idée est ici de voir la variance du committé comme une fonction :

$$\begin{array}{rccc} f: & X & \longrightarrow & \mathbb{R} \\ & \mathbf{x} & \longmapsto & Var(C(\mathbf{x})) \end{array}$$

On peut alors utiliser des algorithmes d'optimisation classiques (que nous détaillerons plus tard) pour converger vers les points de plus grande variance c'est à dire les requêtes que nous souhaitons, de manière similaire à ce que l'on peut voir sur la Figure 3.5.

3.4.2.3 QBC avec synthèse de requêtes et estimateur de variance (QBC3)

Cette variante correspond à peu près à celle décrite précédemment à la différence qu'un modèle est utilisé pour apprendre la variance avant la phase d'optimisation. Elle est similaire à celle utilisée dans l'article de Ren et al. [12].

3.4.3 Description de l'expérimentation menée

3.4.3.1 Description du problème de reférence

Afin de mesurer la performance de nos différentes variantes de QBC, il est nécessaire de les comparer à une référence. Nous avons donc utilisé la méthode classique de l'apprentissage non actif : le tirage aléatoire de points. (Voir 3.1.1). Nous avons décidé d'utiliser un tirage uniforme dans l'espace d'entrée car c'est celui que nous utiliserions si nous devions l'appliquer pour la machine X (on fait l'hypothèse qu'aucune zone de l'espace d'entrée est plus informative que l'autre).

3.4.3.2 Fonctions à apprendre utilisées

Nous avons testé nos méthodes d'apprentissage sur des fonctions, en une et deux dimensions.

3.4.3.2.1 En dimension 1. Nous avons utilisé une fonction de segments linéaires (*changing linear 1d*).

$$\begin{array}{rccc} f: & X & \longrightarrow & \mathbb{R}^1 \\ & \mathbf{x} & \longmapsto & f(\mathbf{x}) \end{array}$$

avec

$$f(x) = \begin{cases} 0.5x, \ si \ x \leq -2 \\ 0.25x, \ si \ -2 < x \leq 0 \\ -0.25x, \ si \ 0 < x \leq 5 \\ 0.5x, \ si \ x > 5 \end{cases} \tag{3.1}$$

La figure 3.6 représente le graphe de la fonction.

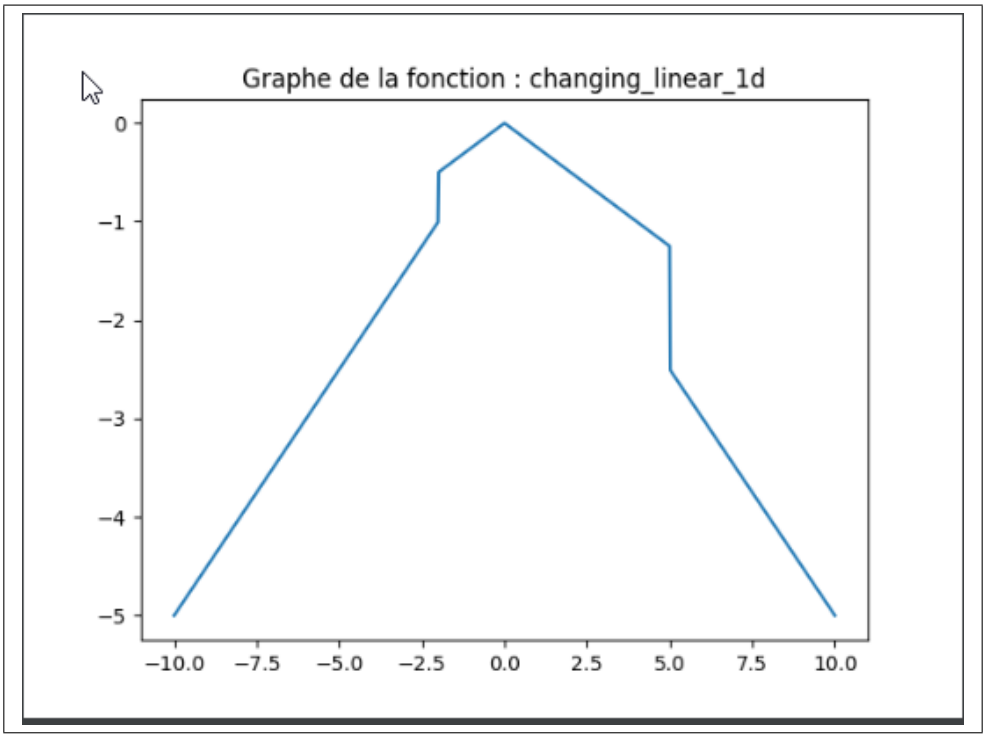

FIGURE 3.6

Nous avons choisis cette fonction car malgré sa simplicité, le fait qu'elle présente des discontinuités rends typiquement son apprentissage non trivial. De plus, ces discontinuités peuvent exister avec des systèmes dynamiques réels.

3.4.3.2.2 En dimension 2. La fonction choisie en deux dimensions est une fonction d'Ackley modifiée, que l'on a multipliée par -1 et sur laquelle ont été rajouté des discontinuités pour la rendre plus complexe (Figure 3.7). Les discontinuités permettent ici aussi de représenter d'éventuels changements de régimes qui peuvent exister avec des systèmes dynamiques réels.

$$\begin{aligned} Ackley_{2D}(x_1, x_2) = -20 * exp(-0.2\sqrt{\frac{1}{2}(x_1^2 + x_2^2)}) \\ -exp(\frac{1}{2}(cos(2\pi x_1) + cos(2\pi x_2)) \\ +e + 20 \end{aligned} \tag{3.2}$$

$$f(x_1, x_2) = \left\{ \begin{array}{llll} -Ackley_{2D}(x_1, x_2) + 8 & x_1 > -1 & x_2 < 1 \\ -Ackley_{2D}(x_1, x_2) + 10 & x_1 > -1 & x_2 \geq 1 \\ -Ackley_{2D}(x_1, x_2) + 12 & x_1 \leq -1 & x_2 < 1 \\ -Ackley_{2D}(x_1, x_2) + 14 & x_1 \leq -1 & x_2 \geq 1 \end{array} \right\} \tag{3.3}$$

On suppose aussi que les paramètres sont définis tels que $x_i \in [-3, 3]$, i={1,2}.

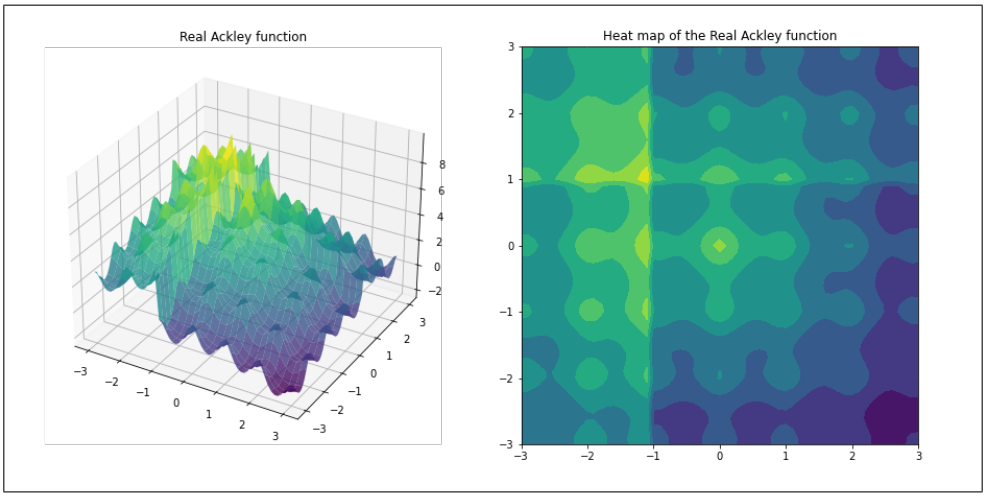

FIGURE 3.7 Graphe de la fonction d'Ackley modifiée

3.4.3.3 Fonction de perte

La fonction de perte (Voir 3.1.2) que nous avons utilisé est l'erreur moyenne quadratique *Mean Square Error (MSE)*.
Pour une base de donnée $D_n = \{(\mathbf{x_1}, \mathbf{y_1}), (\mathbf{x_2}, \mathbf{y_2}), ..., (\mathbf{x_n}, \mathbf{y_n})\}$, elle se calcule comme suit :

$$\begin{array}{rccc} l: & Y^2 & \longrightarrow & \mathbb{R} \\ & (\hat{\mathbf{y}}, \mathbf{y}) & \longmapsto & l(\hat{\mathbf{y}}, \mathbf{y}) \end{array}$$

avec

$$l(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} (\hat{\mathbf{y}} - \mathbf{y})^2$$

Cette fonction de perte est la plus classique dans les problèmes de régression. En effet, elle est quadratique ce qui facilite le calcul du gradient [8] et est optimale dans l'hypothèse où l'erreur aléatoire des sorties est gaussienne ce qui est celle par défaut sans information supplémentaire.

3.4.3.4 Paramètres des différentes variantes

3.4.3.4.1 Présentation des paramètres :

Certains paramètres sont communs à toutes les variantes du QBC.

- **Nombre de membres du comité** : Le nombre de modèles entraînés en parallèle, formant le comité
- **Modèle** : Le type de modèle utilisé pour chacun des membres du comité.
- **Nombre initial d'instances** : Correspond au nombre d'observation initiales avec lesquelles commencer l'apprentissage.
- **Nombre d'instances** : Correspond au nombre de requêtes à générer à chaque itération de l'algorithme.
- **Nombre d'itérations** : Correspond au nombre d'itérations désiré de l'algorithme.

D'autres paramètres sont spécifiques à un type de modèle. Par exemple, pour un modèle de type Multi Layer Perceptron (MLP) on aura les paramètres suivants :

- **Nombre de couches** : Le nombre de couches composant le MLP
- **Nombre de neurones**/ Le nombre, qui peut être différent pour chaque couche, de neurones qui la composent
- **Nombre d'epochs** : Correspond au nombre d'itération de l'algorithme d'apprentissage des paramètres du modèle (cas typique : backpropagation)
- **Batch size** : Correspond au nombre d'exemples qui sont "montrés" à l'algorithme d'apprentissage des paramètres avant que les poids/paramètres du modèle soient recalculés.
- **Patience** : Correspond au nombre d'epochs sans baisse de fonction de perte avant que l'algorithme d'apprentissage des paramètres ne stoppe son apprentissage.

Pour les deux méthodes nécessitant de résoudre un problème d'optimisation, à savoir QBC2 et QBC3, il faut choisir un algorithme d'optimisation (ex : "Nelder-Meads").

Enfin, pour la variante QB3, il faut définir le type de modèle qui va apprendre la variance (l'expert de variance : voir 3.4.2.3). Il faut donc également définir les paramètres du modèle choisi. Dans notre cas, l'expert de variance est un MLP, il faut donc également définir son nombre de couches, le nombre de neurones par couche etc...

3.4.3.4.2 Modèle utilisé pour les membre du comité :

Pour apprendre les fonctions en une et deux dimensions, nous avons choisi d'utiliser un MLP. Le choix de ce type de modèle est motivé par le fait qu'il est générique pour les problèmes de régression et qu'il est par ailleurs très flexible, c'est à dire capable de prédire sans biais. C'est

un réseau de neurones composé de plusieurs couches entièrement connectées entre elles. Lors des expérimentations, le MLP était composé de 5 couches cachées respectivement composées de 128, 256, 256, 256 et 128 neurones. Lors de l'entraînement de ce dernier, un *batch size* de 16 *epochs* a été utilisé. De plus, on utilise de l'*Early stopping* pour arrêter l'entraînement si la fonction de perte ne décroit plus après plusieurs *epochs* consécutives.

3.4.3.4.3 Modèle utilisé pour l'appretissage de la variance :

Le modèle utilisé pour l'apprentissage de la variance est également un MLP. Il est composé de 6 couches cachées respectivement composées de 128, 256, 256, 256, 128 et 64 neurones

3.4.3.4.4 Algorithmes utilisés pour l'optimisation :

Les méthodes de QBC2 et 3 utilisent un algorithme d'optimisation pour trouver les maximums locaux de la variance. Deux algorithmes différents ont été utilisés : *Nelder-Mead* et *ADAM*. Pour le premier, l'implémentation de scipy.optimize.minimize(method="Nelder-Mead") a été utilisée [7]. Cet algorithme est une des façon les plus simples de trouver le minimum d'une fonction, car il nécessite seulement d'évaluer la fonction et non son gradient. De ce fait, il est facile à mettre en place mais est aussi généralement plus long à converger car il a besoin de plus d'itérations que des méthodes avec gradient. La deuxième méthode est *ADAM* [10], une version de la descente de gradient stochastique qui est très utilisée dans le domaine de l'apprentissage profond. "Le nom ADAM est dérivé de l'estimation adaptative du moment" puisque la méthode utilise l'estimation du premier et second moment du gradient pour adapter la valeur du *Taux d'apprentissage* (ou *"learning rate"*). L'avantage de cet algorithme est qu'il converge très rapidement vers un bon résultat et nécessite peu de réglage des paramètres. Cet algorithme est généralement utilisé pour minimiser des fonctions de perte lors d'apprentissage de réseaux de neurones. Nous avons nous-même implémenté ADAM en modifiant certaines fonctions de la librairie tensorflow [1].

Le défi de la minimisation de la fonction de variance est que celle-ci n'est pas toujours continue car elle est calculée à partir de la prédiction des modèles du comité. De ce fait, les algorithmes de minimisation ont plus de mal à converger vers une solution. Afin de tester les performances des 2 algorithmes, des tests sur un MLP entraîné à représenter une fonction dérivée de la fonction d'Ackley sont réalisés. On remarque qu'en 2D, les résultats des deux algorithmes sont bons (Voir Figure 3.9). On remarque aussi d'après la Figure 3.8 qu'*ADAM* est globalement plus lent que *Nelder-Mead* et l'écart augmentant de plus en plus avec le nombre de minimisations consécutives effectuées. Ceci est certainement dû au fait que les méthodes implémentés dans *scipy* sont très bien optimisés. De plus, concernant la précision

des résultats, *Nelder-Mead* semble plus précis en convergeant plus proche du maximum local théoriques.

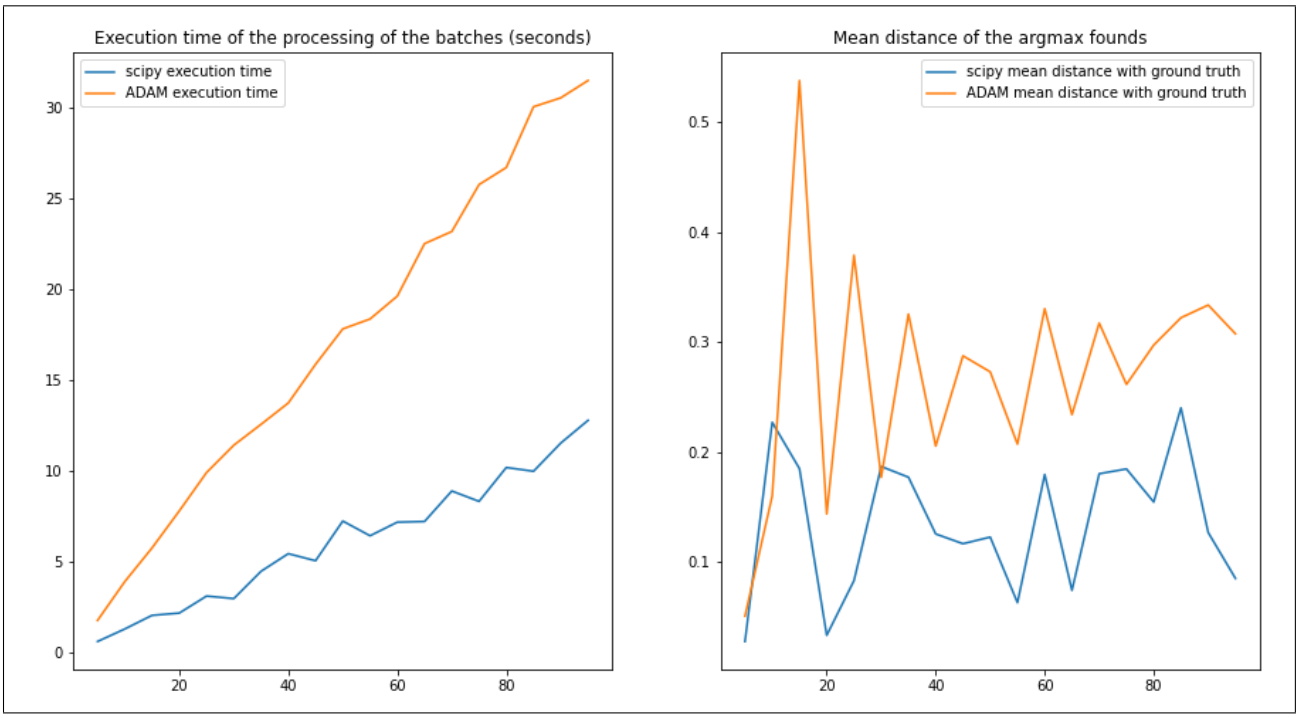

FIGURE 3.8 Gauche : Temps d'exécution d'*ADAM* et de *Nelder-Mead (Scipy)* en fonction du nombre d'éléments à minimiser. Droite : La distance moyenne entre les coordonnées des maximums trouvés et les véritables maximums théoriques

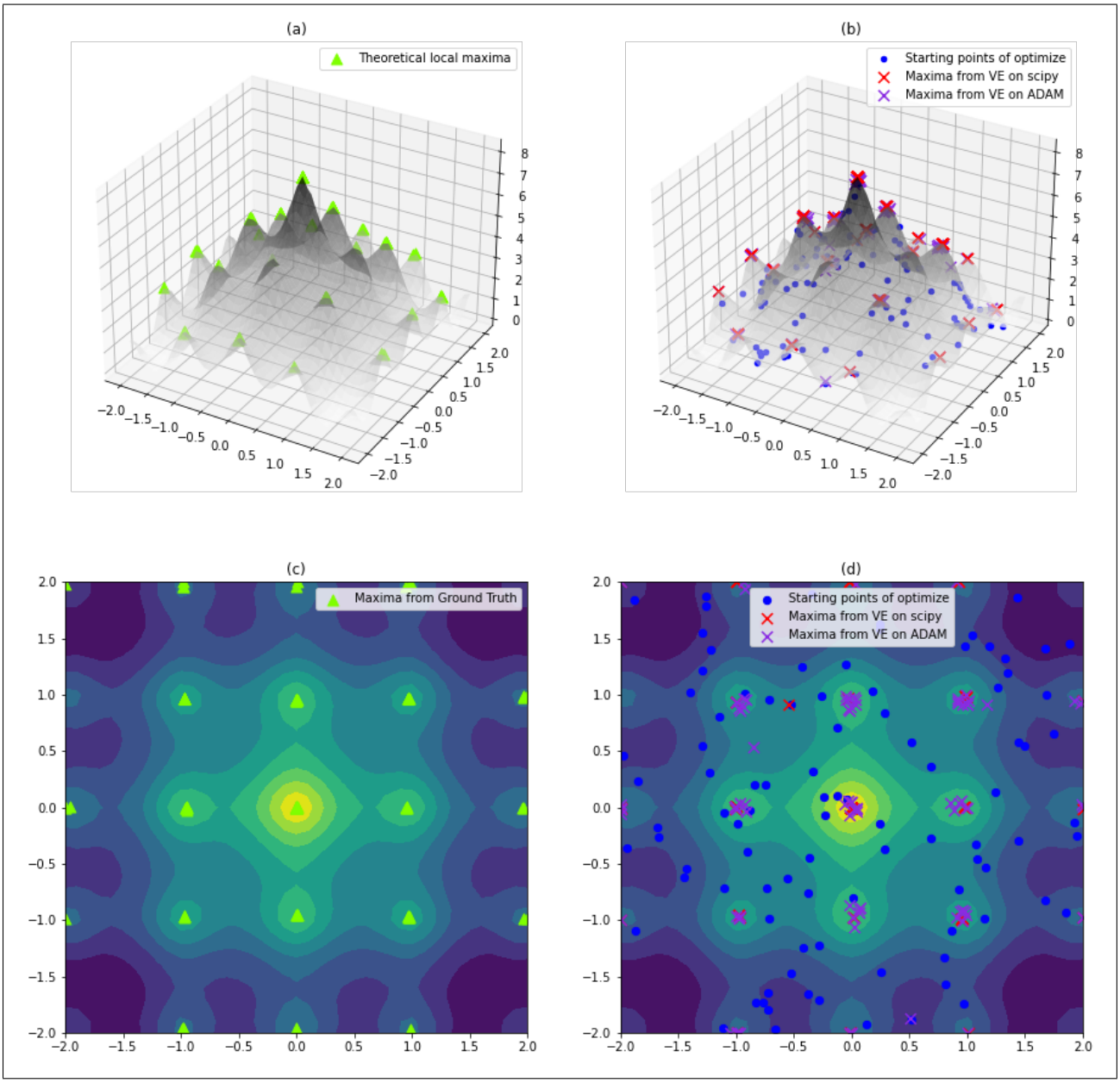

FIGURE 3.9 Résultats de la minimisation de Nelder-Mead et d'ADAM à partir de 100 points d'initialisation différents et effectué sur un MLP composé de 10 couches comprenant chacune 100 neurones.
(a, c) Représentent les résultats théoriques sensés être obtenus.
(b, d) Représentent les résultats des algorithmes.

3.4.3.5 Sélection des requêtes parmi celles synthétisés par optimisation

Dans certain cas avec les méthodes **QBC avec synthèse de requêtes** et **QBC avec synthèse de requêtes et expert de variance**, il est possible que l'algorithme de maximisation

de la variance aboutisse à sélectionner les mêmes pics de variance plusieurs fois. On peut donc regrouper ces points ensemble et les considérer comme une seule requête. Ceci est fait en mesurant la distance de tous les points, les uns par rapport aux autres. Ensuite, on considère que si la distance entre deux points est en dessous d'un certain seuil ($\epsilon = 0.01$), alors ils appartiennent à la même requête.

Un problème peut alors survenir dans le cas où le nombre de requêtes résultantes du tri précédents est inférieur au nombre de requêtes désirées. Pour pallier à ce problème, et avoir le nombre de requêtes demandés, on va augmenter le nombre de points "artificiellement". Pour se faire, on utilise une loi normale centré sur les requêtes obtenues et possédant un écart type égal aux distances moyennes entre le point résultant de l'optimisation et ceux de départs. On utilise ces lois normales pour générer autant de valeurs que de valeurs initiale regroupées en un même point. On obtient ainsi un set de N potentielles requêtes différentes.

Finalement, on sélectionne le nombre de requêtes désirées en sélectionnant les points de plus haute variance afin d'obtenir les requêtes finales que l'on envoi au système réel.

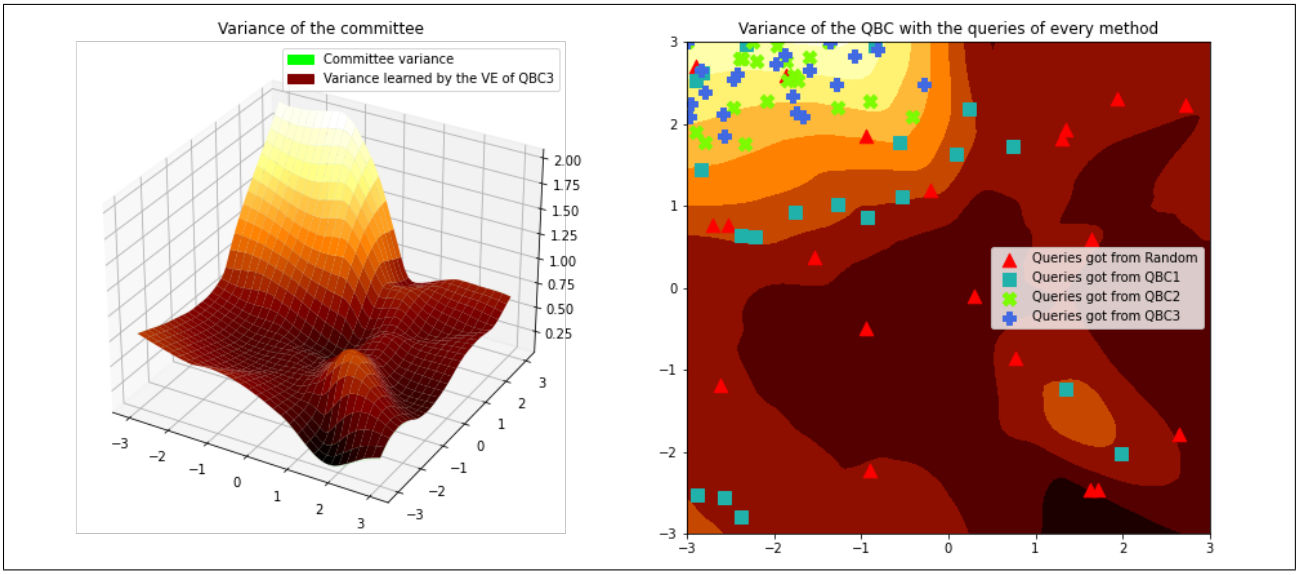

FIGURE 3.10 Requêtes faites par les différentes méthodes sur la fonction de variance associée

3.5 Présentation et analyse des résultats

3.5.1 Une dimension

Les paramètres utilisés dans cette expérience sont disponibles en annexe.

La figure 3.12 représente les requêtes obtenues avec les méthodes définies en 3.4.2. Les erreurs

MSE sont calculés sur un échantillonnage de 1000 points de la fonction réelle (figure 3.11).

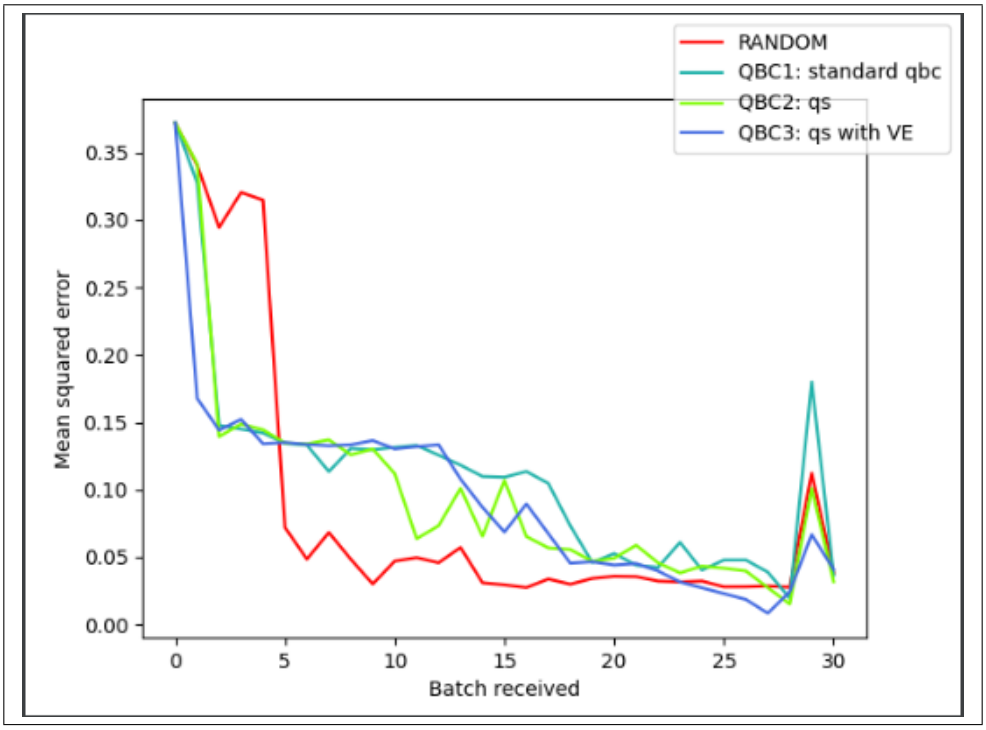

FIGURE 3.11 Moyenne des erreurs aux carré (MSE) calculé à chaque itération

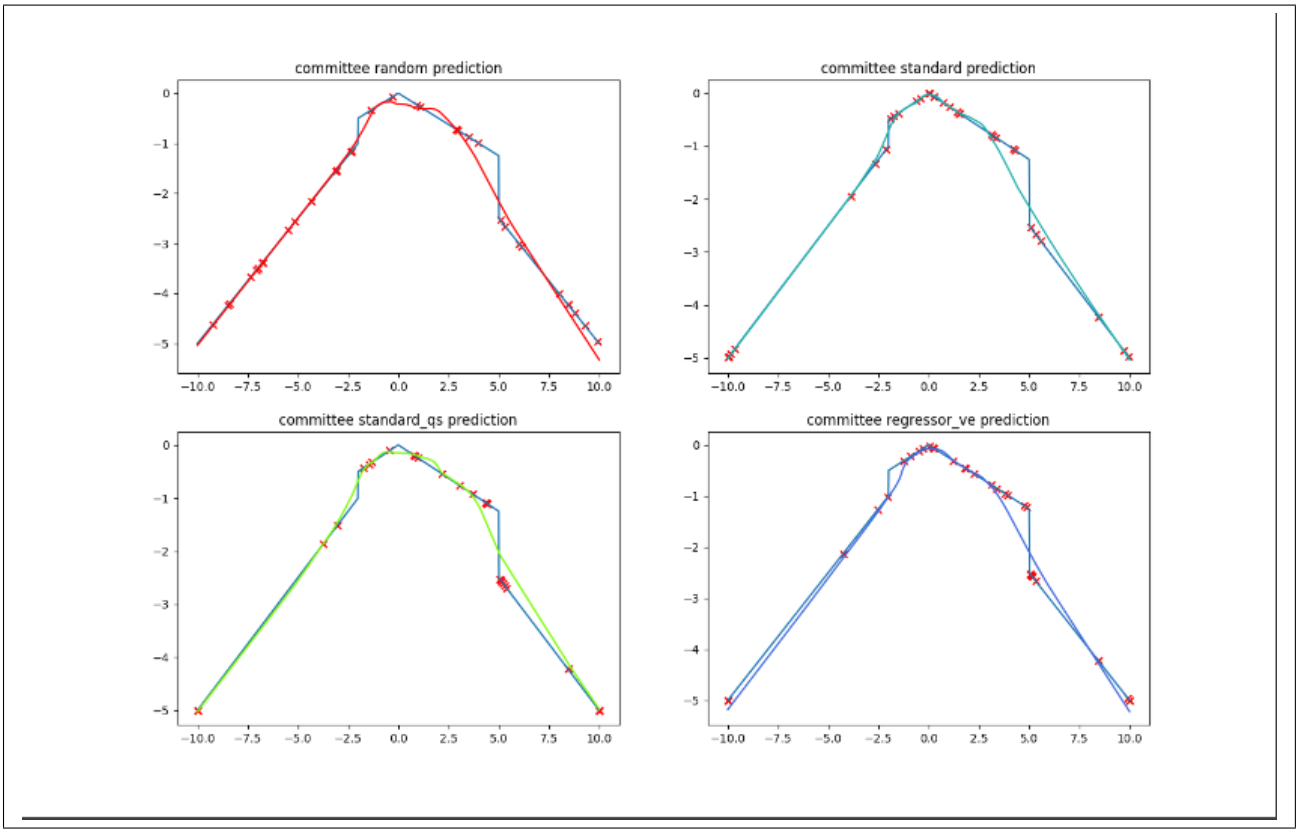

FIGURE 3.12 Requêtes et prédictions obtenues selon les 4 méthodes différentes

On remarque premièrement que les méthodes par QBC donnent des points très localisés sur les discontinuités de la fonction à représenter. Cela est particulièrement marqué avec les méthodes par synthèse de requêtes QBC2 et QBC3. Ceci peut être expliqué par le fait que les discontinuités sont difficiles à apprendre par le modèle étant donné que ces zones possèdent un gradient élevé (théoriquement infini). Le fait de pouvoir repérer ainsi les discontinuités et points critiques est souhaitable car cela pourrait nous permettre de savoir comment scinder l'espace d'entrée afin d'attribuer à chacun des espaces résultant un sous modèle plus adapté à celui-ci. On pourrait alors considerer le modèle de la machine X comme un ensemble de sous-modèle, chacun spécialisé dans sa zone de l'espace.
Concernant les prédictions des différents comités, elles sont de performance semblable, même si le sommet de la courbe semble mieux représenté par les méthodes actives.

Le gain en terme de MSE des méthodes actives par rapport à la méthode aléatoire n'est pas clairement visible. On observe d'abord une phase ou les méthodes QBC progressent très rapidement mais elles sont rattrapées par la méthode aléatoire qui les dépasse dans une seconde phase avant que, finalement, leurs performances ne redeviennent similaire.

Ce phénomène pourrait s'expliquer par le fait que les méthodes actives vont très vite tomber dans une phase d'exploitation, c'est à dire se concentrer dans les zones problématiques alors que la méthode aléatoire va elle continuer d'explorer.

3.5.2 Deux dimensions

Dans cette expérience, nous avons utilisé les mêmes paramètres que pour l'expérience en une dimension excepté :
Nombre initial d'instances = 400
Nombre d'instances = 8
Nombre d'itérations = 50

La figure 3.14 représente les requêtes obtenues avec les méthodes définies en 3.4.2. Les erreurs MSE sont calculés sur un quadrillage de l'espace d'entrée avec un pas de 0.1.(figure 3.13).

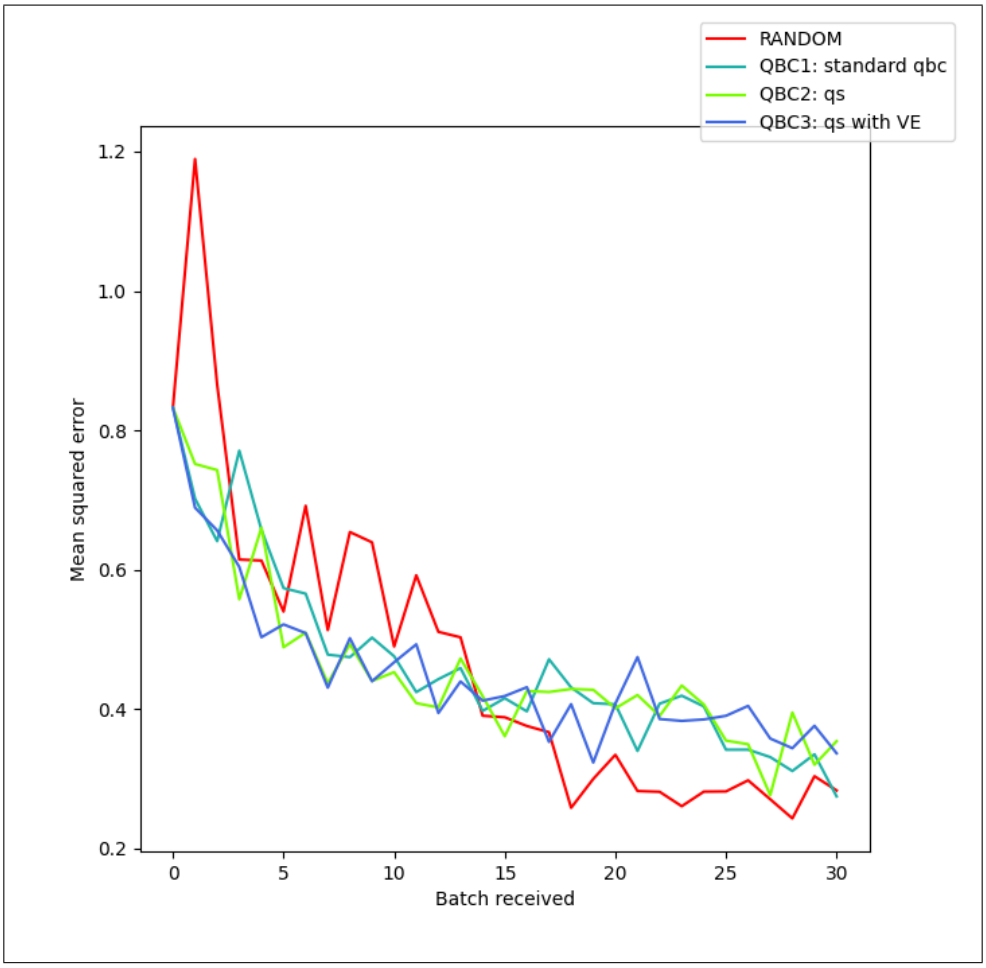

FIGURE 3.13 Moyenne des erreurs aux carré (MSE) calculé à chaque itération

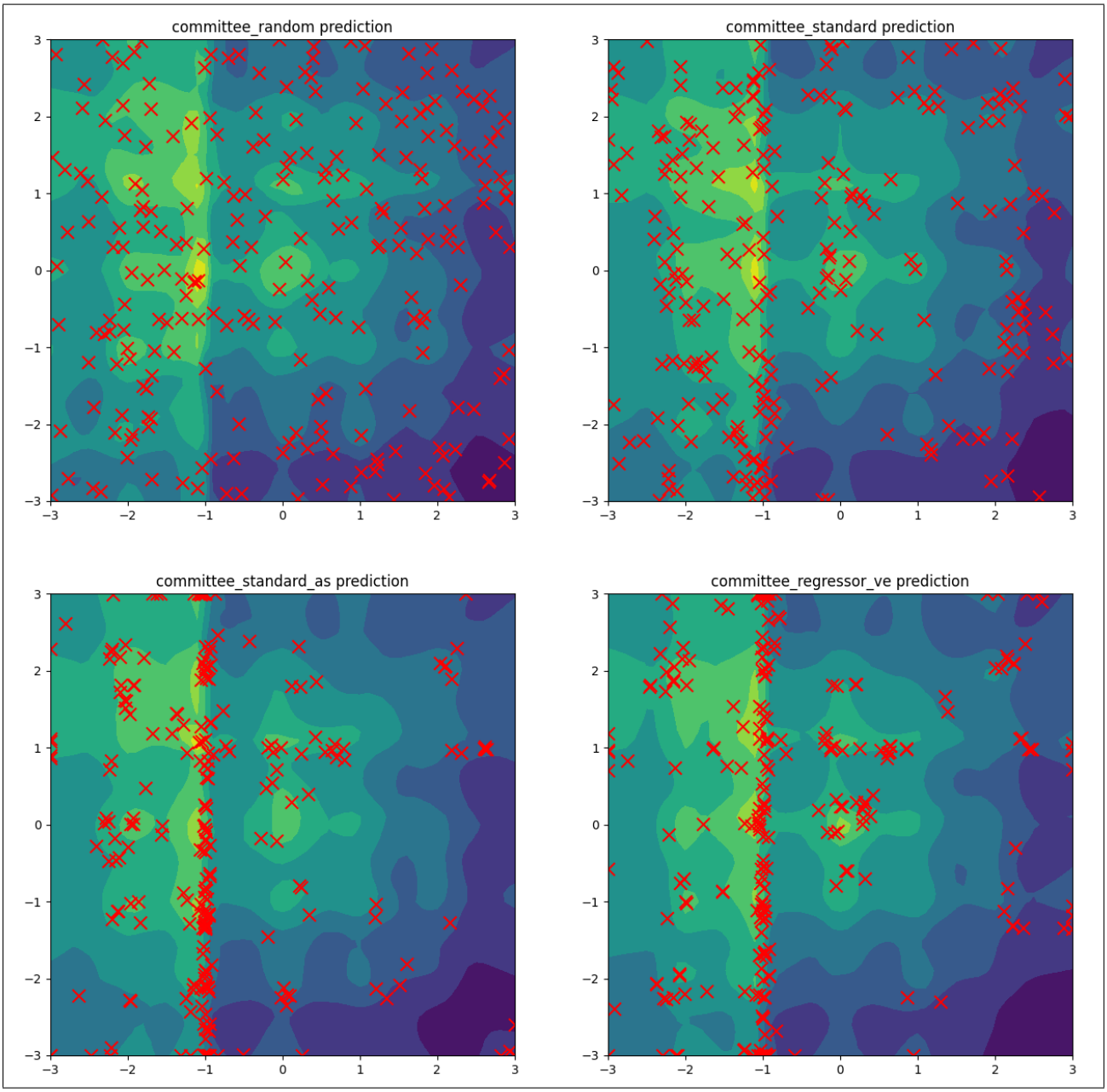

FIGURE 3.14 Requêtes et prédictions obtenues selon les 4 méthodes différentes

On observe ici aussi des points localisés sur les discontinuités. Là encore en observant la figure 3.13 on remarque que les methodes actives commencent par mieux performer que l'aléatoire durant les premières itérations avant que ce ne soit l'inverse lors des suivantes.

3.5.3 L'influence des hyperparamètres de QBC

Afin d'évaluer empiriquement les performances des différentes variantes du QBC, on effectue plusieurs sets de tests avec plusieurs valeurs de nombre d'instances initiales (N_{init}), nombre d'instances (k) et nombre d'itérations (N). On remarque globalement que les 3 méthodes de QBC permettent un meilleur apprentissage que l'aléatoire. En particulier, les méthodes par synthèse de requêtes fonctionnent généralement mieux que la méthode de QBC standard.

— N : (Figure 3.15) Il semble que plus cette valeur augmente et plus la précisions de QBC augmente (plus la valeur du MSE est basse).

— k : (Figure 3.16) Il semble que plus cette valeur est basse et plus le gain entre les QBC et l'aléatoire est grand. Cependant, si l'on baisse k, alors on baisse aussi le nombre d'échantillon récoltés avec le même nombre d'itérations. Ainsi, la précision finale du modèle est meilleure avec un grand k.

— N_{init} : (Figure 3.17) Il semble que pour des petites valeurs de N_{init}, le gain de la méthode par QBC par rapport à l'aléatoire est supérieur. Ce qui semble logique car si N_{init} est petit, on échantillonne plus tôt de meilleures requêtes.

3.5.3.1 L'influence du nombre d'instances N

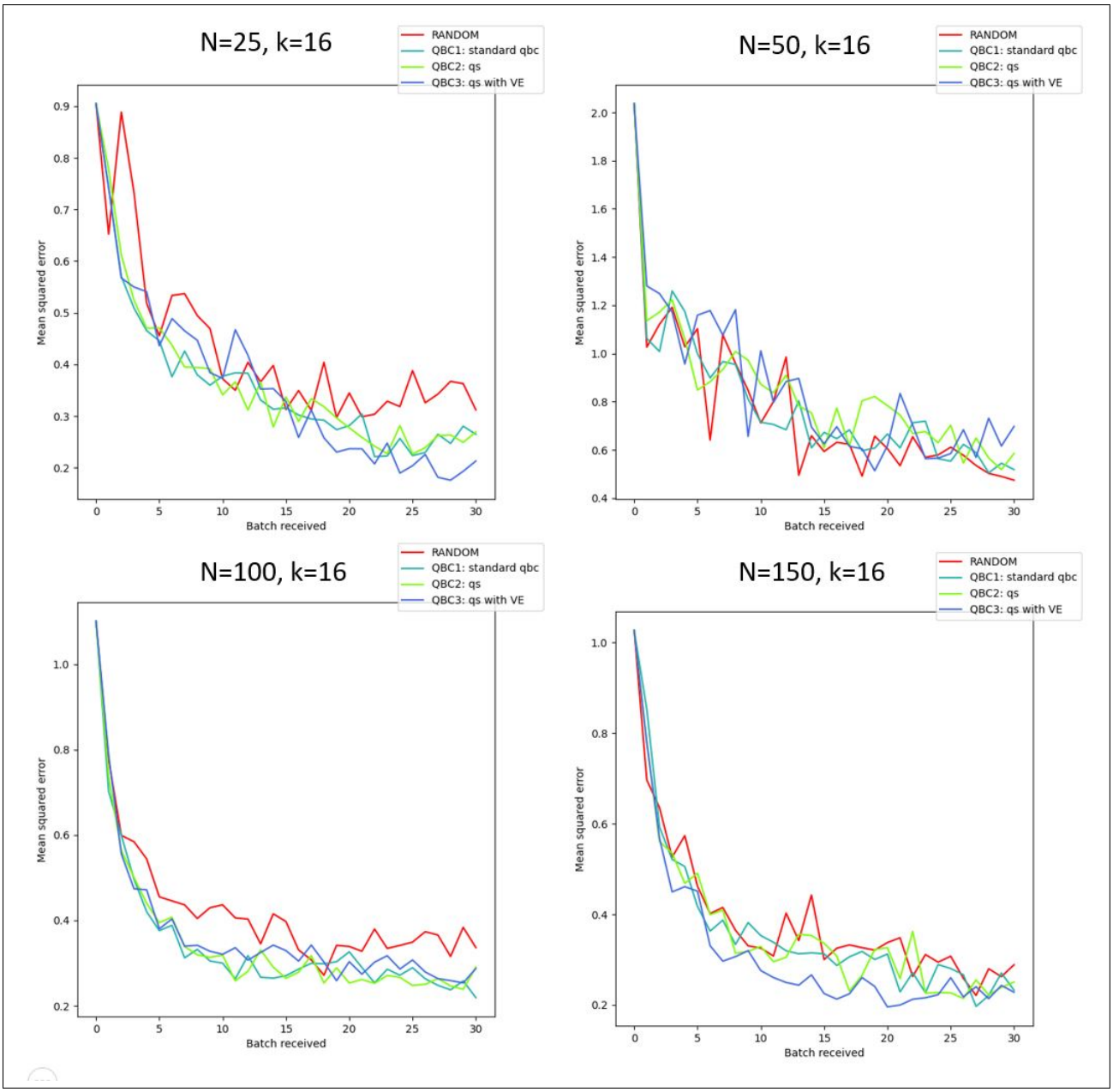

FIGURE 3.15 Fonction de perte *MSE* à chaque itération de l'apprentissage actif. Avec N_{init}=**400**

3.5.3.2 L'influence du nombre de requêtes par itération : k

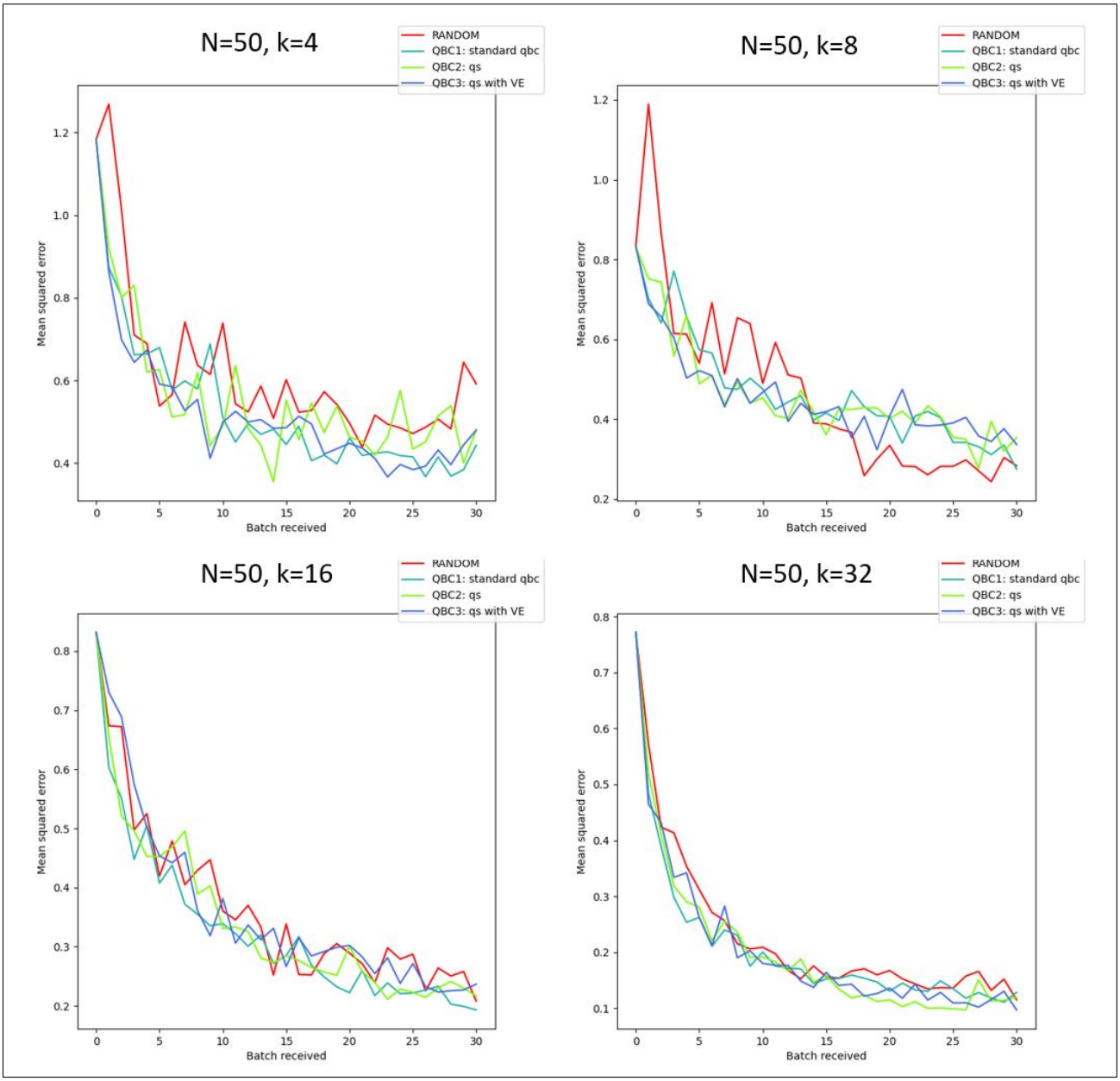

FIGURE 3.16 Fonction de perte *MSE* à chaque itération de l'apprentissage actif. Avec N_{init}=**400**

3.5.3.3 L'influence de la phase de pré-entrainement (nombre d'instances initiales) : N_{init}

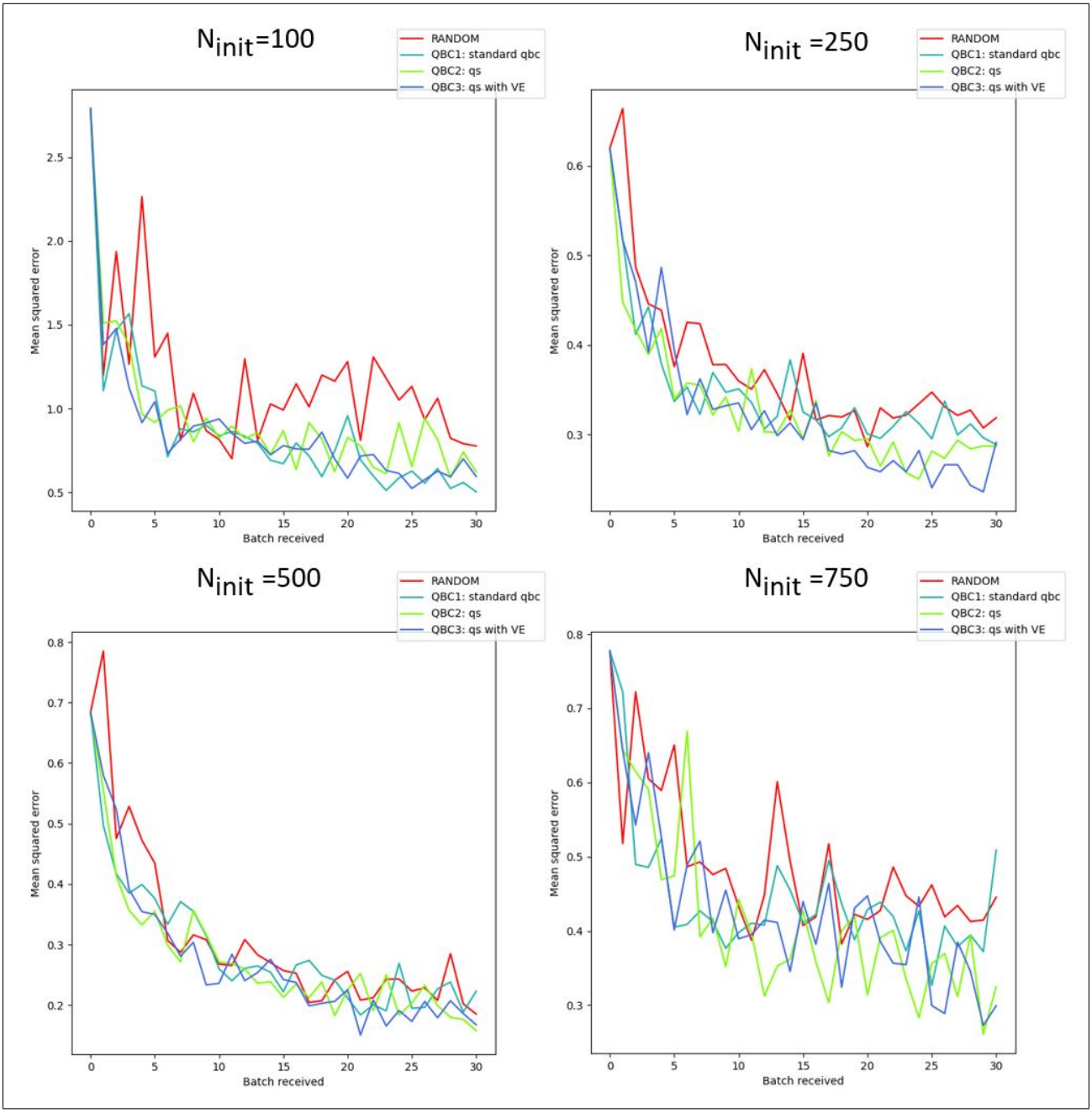

FIGURE 3.17 Fonction de perte *MSE* à chaque itération de l'apprentissage actif. Avec k=**4** et N=**50**

3.6 Discussion et recommandations

Au vu des résultats des expériences en une et deux dimensions, on semble pouvoir tirer quelques conclusions quand à l'efficacité des différentes variantes du QBC. D'abord, ces méthodes permettent de repérer les discontinuités car elles ont tendances à répéter les requêtes dans ces zones. Cette propriété est très utile et représente déjà un gain par rapport à l'aléatoire. Ensuite, les extremum et points critiques sont en général mieux représenté avec les méthodes QBC ce qui peut être intéressant dans une optique de sécurité.

Du point de vue de la performance par rapport à l'aléatoire, deux phases semblent se distinguer. Une première phase qui couvre les premières itérations pendant laquelle les variantes du QBC sont plus performantes dans l'apprentissage (MSE environ 20% moins grand par rapport au comité aléatoire), puis une seconde avec cette fois-ci l'aléatoire qui fourni un meilleure apprentissage. Nous pouvons avancer plusieurs hypothèses pour expliquer ce phénomène.
D'abord, on peut avancer l'explication d'un manque d'exploration de la part des méthodes QBC. En effet, de par leur principe, elles ont tendance à favoriser l'exploitation des zones critiques déjà repéré plutôt que d'aller explorer le reste de l'espace. En petite dimension, le nombre de points critiques est peu important, ce qui fait que les méthodes QBC ont tendance a tirer des points très similaires, donc à peu explorer. Ce problème en est moins un en grande dimension où le nombre de points intéressants devrait toujours être très supérieur au nombre de requêtes. Il faudra confirmer cette hypothèse par une expérience en dimension supérieure. Ensuite, il se peut également que les paramètres internes des méthodes (par exemple l'entraînement de l'expert de variance, l'architecture des celui-ci, l'architecture des membres des comités, nombre de membres etc...) soient adaptés pour les premières itérations mais pas pour les suivantes, ce qui expliquerai la perte de performance. Ainsi, dans une étude future, l'utilisation de paramètres dynamiques par rapport au nombre d'itérations pourrait être prometteuse.

CHAPITRE 4 CONCLUSION

Au cours de ce projet, nous avons souhaité répondre à une problématique d'optimisation de processus industriels à l'aide d'outils issus du domaine de l'apprentissage machine. Plus particulièrement, nous avions deux objectifs distincts liés à l'entraînement de modèles appelés des surrogates. Ces modèles représentent des systèmes réels et permettent de simuler leur utilisation, ce qui est utile lorsque l'utilisation de ces derniers est coûteuse ce qui est notre cas.

4.1 Synthèse des travaux

Notre premier objectif était de développer et d'implémenter une méthode d'apprentissage actif : les *requêtes par votes* (ou *Querry-By-Committe (QBC)*). Nous avons comparé les performances de 3 variantes du QBC dont certaines utilisant la *Synthèse de requête* permettant de générer des requêtes dans l'ensemble de l'espace. Elles semblent bien pouvoir performer au moins aussi bien que la méthode aléatoire. Le QBC est une méthode invariante par rapport au choix du modèle ce qui était souhaitable car cela nous permet de pouvoir changer de modèle à volonté

Notre second objectif était de construire une infrastructure permettant l'entraînement automatisé de modèles surrogates étant capable de communiquer avec la machine. Celle-ci a été baptisée *MOODS ((Model Online Optimization Distributed System)* et a été testée avec succès sur un problème simple du pendule inversé avec un espace d'entrée de dimension 4. Elle n'est pas détaillé dans ce rapport car a été réalisé par mon collègue, stagiaire de L'ETS.

4.2 Limitations de la solution proposée

Si nos méthodes semblent bien fonctionner sur les premières itérations, cela reste à confirmer pour les itérations suivantes. Il n'est pas encore clair si ces contre-performances sont dues à la dimension de test, à un problème fondamental dans la méthode où tout simplement à un manque de dynamisme dans les paramètres. Il faudra explorer ces différentes pistes lors d'expérimentations futures

Par ailleurs, nous n'avons pas pu tester MOODS sur les données issues de la machine X de l'entreprise X, ces derniers n'ayant pas pu nous envoyer des données à temps. Cependant, il serait intéressant de le faire afin d'appliquer MOODS à un système réel.

4.3 Améliorations futures

Les pistes d'améliorations futures seraient d'abord dans une meilleure compréhension de l'impact des différents paramètres des méthodes QBC sur leurs performances d'apprentissage. Cette vision plus claire des variantes du QBC pourrait nous permettre d'implémenter une adaptation dynamique des paramètres par rapport au nombre d'itérations.

RÉFÉRENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow : Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Naoki Abe and Hiroshi Mamitsuka. Query learning strategies using boosting and bagging. In *ICML*, 1998.

[3] D. Cohn, Z. Ghahramani, and M.I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 1996.

[4] Christos Doukas, Dimitrios Chantzis, Panagiotis Stavropoulos, Alexios Papacharalampopoulos, and George Chryssolouris. Monitoring and control of manufacturing processes : A review. volume 8, 06 2013. doi : 10.1016/j.procir.2013.06.127.

[5] Technical Director Lam Research Corporation Dr. Steve Sirard. Introduction to plasma etching, 2018. URL `https://willson.cm.utexas.edu/Teaching/LithoClass2018/Files/Introduction20to20Plasma20Etching_Lecture_110118_Sntzd.pdf`.

[6] Alexander I. J. Forrester, Andras Sobester, and Andy J. Keane. *Engineering Design via Surrogate Modelling - A Practical Guide.* Wiley, 2008. ISBN 978-0-470-06068-1.

[7] Fuchang Gao and Lixing Han. Implementing the nelder-mead simplex algorithm with adaptive parameters. 51(1) :259–277, 2012. ISSN 0926-6003. doi : 10.1007/s10589-010-9329-3. URL `https://doi.org/10.1007/s10589-010-9329-3`.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[9] Hassan K Khalil. *Nonlinear systems ; 3rd ed.* Prentice-Hall, Upper Saddle River, NJ, 2002. URL `https://cds.cern.ch/record/1173048`.

[10] Diederik P. Kingma and Jimmy Ba. Adam : A method for stochastic optimization, 2014. URL `http://arxiv.org/abs/1412.6980`. cite arxiv :1412.6980Comment : Published as

a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[11] Frank Lewis and Draguna Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *Circuits and Systems Magazine, IEEE*, 9 :32 – 50, 01 2009. doi : 10.1109/MCAS.2009.933854.

[12] Simiao Ren, Yang Deng, Willie J. Padilla, and Jordan M. Malof. Hyperparameter-free deep active learning for regression problems via query synthesis. *CoRR*, abs/2201.12632, 2022. URL `https://arxiv.org/abs/2201.12632`.

[13] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[14] Burr Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[15] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. COLT '92, page 287–294, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi : 10.1145/130385.130417. URL `https://doi.org/10.1145/130385.130417`.

[16] Dongrui Wu, Chin-Teng Lin, and Jian Huang. Active learning for regression using greedy sampling. *CoRR*, abs/1808.04245, 2018. URL `http://arxiv.org/abs/1808.04245`.

[17] Christoph Zimmer, Mona Meister, and Duy Nguyen-Tuong. Safe active learning for time-series modeling with gaussian processes. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/b197ffdef2ddc3308584dce7afa3661b-Paper.pdf`.

ANNEXE A PARAMÈTRES UTILISÉS DANS L'EXPÉRIMENTATION EN 1D

nb_members=3

ratio_initial_bootstrap=0.8

model=<class 'moods.qbc.models.MlpEstimator'>

nb_layers=5

nb_neurones=[128, 256, 256, 256, 128]

dim_input=1

dim_output=1

variance_evaluator='MLP'

num_sample_train_VE=None

nb_layers_VE=6

nb_neurones_VE=[128, 256, 256, 256, 128, 64]

epochs=50

batch_size=24

patience=10000

verbose=0

optimizer='scipy'

threshold=0.01

ratio_augmentation=0.5

scaled_min_search=array([-10])

scaled_max_search=array([10])

n_instance_initial=5

random_pool_length=100

n_instance=1

nb_iteration=30