HAL
open science

Apprentissage incrémental avec peu de données pour la reconnaissance de caractères manuscrits en-ligne

Abdullah Almousa Almaksour, Harold Mouchère, Eric Anquetil

To cite this version:

Abdullah Almousa Almaksour, Harold Mouchère, Eric Anquetil. Apprentissage incrémental avec peu de données pour la reconnaissance de caractères manuscrits en-ligne. Traitement du Signal, 2009, pp.00. hal-00491329

HAL Id: hal-00491329

https://hal.science/hal-00491329v1

Submitted on 11 Jun 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Apprentissage incrémental avec peu de données pour la reconnaissance de caractères manuscrits en-ligne

Incremental learning with few data for online handwritten character Recognition

Abdullah Almaksour[1], Harold Mouchère[2], Eric Anquetil[1]

[1]Laboratoire IRISA/INSA
Campus Universitaire de Beaulieu
Avenue du Général Leclerc
35042 RENNES Cedex

[2]Laboratoire IRCCyN
Rue Christian Pauc
BP 50609
44306 Nantes Cedex 03
{Abdullah.Almaksour, Eric.Anquetil}@irisa.fr
Harold.Mouchere@univ-nantes.fr

Résumé

Dans ce papier, nous présentons un nouvel algorithme d'apprentissage incrémental d'un système de reconnaissance en-ligne de caractères manuscrits. L'objectif est d'apprendre « à la volée » toute nouvelle classe de caractères à partir de très peu d'exemples de caractères tout en optimisant les classes déjà modélisées au fur et à mesure de la saisie de nouveaux exemples. Le système proposé est capable de surmonter le problème du manque de données d'apprentissage lors de l'introduction d'une nouvelle classe de caractères grâce à la synthèse de caractères artificiels. Les tests ont été conduits dans le cadre d'un apprentissage incrémental mono-scripteur de lettres minuscules cursives sur une base de 18 scripteurs. Les résultats montrent qu'un bon taux de reconnaissance (environ 90%) est atteint en utilisant seulement 5 exemples d'apprentissage par classe. De plus, ce taux augmente rapidement pour atteindre 94% pour 10 exemples, et environ 97% pour 30. Une réduction d'erreur de 40% est obtenue en utilisant la synthèse de caractères par rapport à une stratégie sans synthèse.

Mots clés : Apprentissage incrémental, reconnaissance de caractères manuscrits en-ligne, synthèse de données manuscrites, rejet d'ambiguïté, systèmes d'inférence floue.

Abstract

1 Introduction

In the last decade, the world witnessed the emergence of touch screen devices, from Personal Digital Assistants (PDAs) to Tablet PCs and other instruments that use pen-based interfaces. More and more efforts are needed to make online handwriting recognition systems more robust and adaptive in order to meet the increasing user requirements. One of these new requirements is to build a recognition system that allows the user to choose his own group of gestures and assign them to different interactive commands, e.g. "copy", "paste", "undo", etc. This application context imposes specific conditions on the used learning approach. In such approach, the system must be able to rapidly learn a new unseen gesture using only few data. The later is due to the fact that users would rarely be willing to wait and repeat each new gesture a dozen of times for training the system. Furthermore,

such a learning process must be progressively repeated for each newly added data. For all these reasons, this study aims at introducing a fast incremental learning strategy for online handwriting recognition systems. The main difficulty is to build on-the-fly a handwriting classifier from scratch, with the constraint of few available learning examples, and then to adapt it incrementally in order to achieve high recognition rate as soon as possible. In this paper, we propose a new algorithm for structure and parameters incremental learning with a partial instance memory. This system is used in our context for the recognition of online handwritten characters, and is able to learn a new class while continuing to evolve, example after example, without using all past data.

2 Fuzzy inference system

We have chosen a Fuzzy Inference System (FIS) as a classification approach for several reasons: first of all, it is a light system, with few of parameters. And also, its flexible nature allows to adapt it easily to absorb a new knowledge. The classifier is formalized by a zero-order Takagi-Sugeno FIS. The fuzzy rules make a link between intrinsic models that describe the properties of the handwritten characters and the corresponding label. Each intrinsic model is defined by a set of fuzzy prototypes in n dimensions. A fuzzy rule is built for each fuzzy prototype:

$$\textbf{IF } X \text{ est } P_r \textbf{ THEN } s_1^r = a_1^r \textbf{ ... and } s_c^r = a_c^r \textbf{ ... and } s_C^r = a_C^r$$

As each prototype can participate to the description of each class, the rule has numeric conclusions connecting the prototype with each class by a score. Fuzzy prototypes are defined by their membership function (see equation 1). This one is represented by a hyper-ellipsoidal Radial Basis Function (RBF) characterized by a center and a covariance matrix. To recognize an unknown character X, its membership degree to all fuzzy prototypes is computed and the *sum-product* inference is used to compute the system output scores for each class.

3 Incremental learning principals

There are two manners by which a prototype-based recognition system can be incrementally learned. The first manner is by creating new prototypes for newly available data. The second is by modifying the existing prototypes according to new data. Although prototype creation helps to improve significantly the classifier capacity (recognition rate), the complexity of the classifier (computing time and memory space) becomes unacceptable if prototypes number becomes sizable. For this reason, the use of prototypes adjustment is essential in order to get a dynamic and light system. Prototypes adjustment (or adaptation) can improve the capacity with same complexity of the classifier, but this technique has some limitations. It is difficult for it to change rapidly the decision borders in the case where the new knowledge (presented by the new data) is little represented by the existing prototypes. In such case, the decision of creating a new prototype is intuitively more reasonable. We note that both techniques are complementary, and the first challenge to have an efficient incremental learning model is to find the best strategy to combine these two techniques.

In non-incremental learning systems, where the system is built using learning dataset with sufficient number of examples, supervised or unsupervised clustering methods can be used to calculate the prototypes (centers and covariance matrices) for each class. In our context, when adding a new rule, a hyper-spherical prototype is creating in the feature space using the only one current example. A first simple solution is to initialize the prototype's center by the current example, and the covariance matrix by a matrix proportional to identity matrix (thus setting the initial size of the prototype). These prototypes are then distorted to have hyper-ellipsoidal shape thanks to the adjustment technique.

For prototype adaptation, we use the ADAPT method, which allows to modify all the prototypes of FIS by re-centering and re-shaping them for each new example. This is done according to their participation in the recognition process. The principle of ADAPT is inspired by the LVQ algorithm. The method is detailed in the paper.

4 Incremental learning strategies

We present and compare in this paper two different incremental learning strategies. The first one consists of two distinct phases. During the first phase, that we called fast incremental learning phase, a significant number of prototypes are added in order to rapidly enhance the classifier's performance. Then in the second phase, called adaptation phase, prototype creation is minimized and a classifier adaptation technique (ADAPT) is then used to continue the learning process.

The second strategy is based on the detection of a confusion reject in order to trigger the new prototype creation process. Using this strategy, we aim to limit prototypes number in the system by avoiding creating prototypes in places where there is no likelihood of confusion. This eliminates adding "unnecessary" prototypes for a specific class where this class was already dominant. The advantage of the confusion-driven strategy is that for "easy to learn" classes, prototype creation will be early limited by the absence of confusions with other classes. However, the system can continue to add new prototypes in order to learn "difficult to learn" classes as long as confusions are triggered. Confusion reject can be realized by defining a reject zone on each side of decision boundaries. Each pattern within one of these zones is considered as potential error and is therefore rejected. Figure 1 shows an example in two dimensions of confusion reject zones for three classes.

5 Incremental learning acceleration using artificial data generation

Due to lack of knowledge in the beginning of learning a new class, generating synthetic handwritten characters can improve the quality of created prototypes to be more representative of user writing style. Thus, a new prototype is created by calculating the center and the covariance matrix from synthetic characters, which results in hyper-ellipsoidal prototypes instead of hyper-spherical prototypes (created using the original character only). We use in our system two techniques of characters generation. The first one uses classical image distortions, such as scaling and slanting. The second one is based on the particularity of on-line handwriting; it applies two on-line distortions

on the character: speed variation and curvature modification. Thus, several variations of the same class are generated from a single example of this class. Distortions limits are carefully chosen in order to avoid generating an artificial character that does not look like any more the original one, so it will not help to learn the user writing style.

6 Results and conclusion

The experiments are based on the recognition of the 26 isolated lower case Latin letter. The writer specific datasets were written on a PDA by 18 writers. Each writer has randomly inputted 40 times each character, i.e 1040 characters per writer. In order to estimate the performance of the incremental learning strategy for each writer, we proceed by a 4-fold cross-validation technique. Three quarters of the dataset (780 letters) are used to incrementally learn the system, and one quarter (260 letters) is used to estimate the evolving of system capacity during the learning process. The presented results in the figures are the average of the 18 tests (18 writers). Each pattern in our system is described by a set of 21 features. A new example of each class is presented to the system in each learning cycle.

We compare in these experiments the performance of the two incremental learning strategies in terms of the complexity of the classifier, and the quality of the classifier. we evaluate also the impact of using the artificial characters generation on the quality and the complexity of the classifier in our incremental learning system.

We note that the confusion-driven strategy results in a classifier with a quality equal to or greater than that obtained with the two-phase strategy, with creating fewer prototypes. We find that using the confusion-driven strategy with artificial characters generation, a recognition rate about 90% is reached after only 5 learning examples, and such rate rapidly improves reaching 94% after 10 examples, and about 97% after 30 examples. We note also that recognition error rate decreases by 40% using artificial characters generation techniques.

We conclude that confusion-driven strategy achieves a better recognition rate for the same number of prototypes comparing to two-phase one. It can be also noted that the quality/complexity ratio of the classifier is enhanced thanks to the artificial characters generation. A special emphasis for a possible future work is placed on reducing the

number of prototypes in the system either by deleting the "useless" prototypes or by merging redundant ones. We plan also to explore other approaches to synthesize handwriting, inspired in particularly by the Sigma-lognormal model.

Keywords : Incremental learning, online handwritten characters recognition, handwriting generation, confusion reject, fuzzy inference systems.

7 Introduction

Avec l'émergence des assistants personnels numériques (PDA) et des téléphones mobiles de nouvelle génération (smartphones) utilisant des interfaces orientées stylo, les performances des systèmes de reconnaissance de caractères en-ligne doivent être optimales. De plus en plus d'efforts sont nécessaires pour rendre ces systèmes plus robustes et adaptables afin de répondre aux nouveaux besoins des utilisateurs. Parmi ces nouveaux besoins, il apparaît intéressant de pouvoir proposer à l'utilisateur de choisir son propre groupe de gestes et de les assigner à différentes commandes interactives, par exemple, « copier », « coller », « annuler », etc. Ce contexte applicatif impose des contraintes spécifiques à la technique d'apprentissage utilisée. L'objectif est d'être capable d'apprendre rapidement en utilisant peu de données. En effet, les utilisateurs sont rarement prêts à saisir chaque nouveau symbole plus d'une douzaine de fois pour l'apprentissage du système. Pour répondre à ces besoins, cette étude propose un algorithme original d'apprentissage incrémental rapide pour les systèmes de reconnaissance d'écriture manuscrite en-ligne. La modélisation repose sur des Systèmes d'Inférence Floue (SIF) particulièrement bien adaptés à la mise en œuvre d'un apprentissage incrémental. Nous validons ici notre approche sur les lettres minuscules manuscrites, mais l'objectif à long terme est d'utiliser cette même approche dans n'importe quel système de reconnaissance en-ligne de gestes ou de symboles manuscrits.

La difficulté principale de cette approche est de construire « à la volée » un classifieur d'écriture manuscrite, en s'appuyant sur très peu de connaissances disponibles (quelques exemples d'apprentissage), puis de l'adapter

progressivement afin d'atteindre un fort taux de reconnaissance le plus rapidement possible.

Un algorithme d'apprentissage incrémental est défini dans [24] comme répondant aux critères suivants :

1. il doit être capable d'apprendre des connaissances supplémentaires à partir des nouvelles données ;
2. il ne doit pas nécessiter l'accès aux données d'origine (c'est-à-dire les données qui ont été utilisées pour apprendre le classifieur actuel) ;
3. il doit préserver les connaissances déjà acquises ;
4. il doit être en mesure d'apprendre de nouvelles classes susceptibles d'être introduites avec de nouvelles données.

Ces quatre points, qui s'appliquent pour tout problème général de l'apprentissage incrémental, correspondent parfaitement aux caractéristiques particulières de notre problème, un algorithme d'apprentissage rapide pour un système de reconnaissance en-ligne de caractères manuscrits.

Il existe plusieurs algorithmes proposés pour effectuer l'apprentissage incrémental, bien que beaucoup d'entre eux ne répondent pas à tous les critères pour être considérés comme des approches d'apprentissage incrémental.

Nous pouvons distinguer principalement deux types d'algorithmes d'apprentissage incrémental : les algorithmes d'apprentissage incrémental de paramètres et les algorithmes d'apprentissage incrémental de structure et de paramètres. L'apprentissage incrémental de paramètres peut être assimilé à ce que l'on appelle « adaptation ». Dans ces systèmes, la structure est fixe et initialisée au départ et les paramètres du système sont appris progressivement en fonction des données d'apprentissage disponibles. Quelques exemples de ces systèmes sont présentés dans [11], [10] et [18].

La plupart des algorithmes d'apprentissage incrémental de structure et de paramètres sont basés sur le principe de l'algorithme de clustering ART [5], tel que [4], [26], [8] et [15]. Le problème principal de ces systèmes est qu'ils sont sensibles à la sélection du paramètre de vigilance, aux niveaux de bruit dans les données d'apprentissage et à l'ordre dans lequel les données d'apprentissage sont présentées.

Une autre approche que l'on rencontre souvent en apprentissage incrémental s'appuie sur des systèmes à base d'ensemble de classifieurs, comme dans [24] et [17]. Dans ces systèmes, au lieu de créer de nouveaux clusters pour modéliser les nouvelles connaissances, se sont de nouveaux classifieurs qui sont appris et ajoutés au système de façon incrémentale après l'acquisition d'une certaine quantité de données. La performance du système est donc basée sur la performance synergique d'un ensemble de classifieurs faibles. Cette technique peut être considérée comme un processus d'apprentissage incrémental hors-ligne (c'est-à-dire non-instantané), il est souvent utilisé pour apprendre incrémentalement une base de taille importante en opérant une séquence d'apprentissage « batch » sur des sous-ensembles de cette base. Cependant, les classifieurs créés sont très difficilement adaptables après leur apprentissage.

Dans un algorithme d'apprentissage incrémental, la base d'apprentissage n'est pas disponible a priori, puisque les exemples d'apprentissage arrivent au fil du temps. Bien que les systèmes d'apprentissage en-ligne mettent à jour et améliorent constamment leurs modèles, ils ne sont pas forcement tous basés sur une approche incrémentale. Dans certains systèmes, le modèle est complètement reconstruit à chaque étape d'apprentissage incrémental en utilisant l'ensemble des données disponibles, ce sont des systèmes avec mémoire d'exemples complète (full instance memory) [25] . Par contre, si l'algorithme d'apprentissage modifie le modèle uniquement en fonction du dernier exemple d'apprentissage, c'est un algorithme incrémental sans mémoire d'exemples (no instance memory) [14] [13]. Une troisième approche est celle des systèmes à mémoire partielle d'exemples, qui sélectionnent et gardent un sous-ensemble réduit d'exemples pour les utiliser dans les prochaines étapes d'apprentissage [1].

Dans ces travaux, on constate que les modèles à base de prototypes sont généralement les mieux adaptés aux problèmes d'apprentissage incrémental rapide, où le modèle doit être modifié pour chaque nouvel exemple sans disposer d'une mémoire complète des exemples précédents.

Dans ce papier, nous proposons un nouvel algorithme d'apprentissage incrémental de structure et de paramètres avec une mémoire partielle d'exemples. Ce système est utilisé dans notre contexte pour la reconnais-

sance en-ligne de caractères manuscrits, et il est capable d'apprendre une nouvelle classe tout en continuant à évoluer, exemple après exemple, sans utiliser de données antérieures. Afin de surmonter le problème du manque d'exemples, et d'augmenter la vitesse d'apprentissage, en améliorant rapidement le taux de reconnaissance du système, nous intégrons dans le processus d'apprentissage des techniques originales de synthèse de caractères manuscrits, présentés dans nos précédents travaux [20].

La suite de cet article est organisé comme suit. Dans la section 8, nous présentons la méthode de classification basée sur des Systèmes d'Inférence Floue (SIF) et les principes de base de notre système d'apprentissage incrémental. Ensuite, nous décrivons dans la section 9 deux stratégies différentes d'apprentissage incrémental. La section 10 expose alors la méthode que nous avons utilisée pour générer des caractères artificiels afin d'accélérer le processus d'apprentissage. Enfin la section 11 présente les résultats de nos expérimentations.

8 Principes généraux de l'approche proposée

8.1 Les Systèmes d'Inférence Floue

Les Systèmes d'Inférence Floue (SIF) étendent les principes des systèmes à base de règles classiques en modélisant les imperfections liées aux connaissances manipulées grâce aux outils de la théorie des sous-ensembles flous. Les mécanismes de raisonnement en résultant sont ainsi plus robustes et plus proches de la réalité [2].

Le SIF que nous utilisons est du type Takagi-Sugeno d'ordre 0 constitué de N règles. Chaque règle R_r est composée d'une prémisse et d'une conclusion. La prémisse correspond à une modélisation intrinsèque d'une classe ou d'une partie d'une classe par un prototype flou P_r défini dans l'espace des caractéristiques E. La conclusion associe à chaque prototype son degré d'appartenance s_c^r à chaque classe c. Dans un problème à C classes, les règles s'écrivent donc :

$$\textbf{SI } X \text{ est } P_r \textbf{ ALORS } s_1^r = a_1^r \textbf{ ... et } s_c^r = a_c^r \textbf{ ... et } s_C^r = a_C^r$$

avec X la forme à reconnaître dans E, P_r sont les prototypes flous et les a_c^r des

valeurs constantes (ordre 0). Dans ce papier, la notion de prototype est défini à la fois par son centre μ_r et sa zone d'influence représentée par une fonction à base radiale hyper-ellipsoïdale de centre μ_r et dont la forme est donnée par la matrice de covariance Q_r. Le degré d'appartenance d'une forme X utilise une fonction de Cauchy basée sur la distance de Mahalanobis $d_{Q_r}(X, \mu_r)$:

$$\beta_r(X) = \frac{1}{1 + d_{Q_r}(X, \mu_r)} \ . \tag{1}$$

Les prototypes flous sont appris séparément sur chaque classe en utilisant un algorithme de classification non supervisée basée sur la méthode de c-moyenne possibiliste. Les conclusions de chaque règle peuvent être calculées avec la méthode pseudo-inverse (les moindres carrés récursif). Elle permet d'optimiser la discrimination des classes.

Pour déterminer la classe d'une forme inconnue X, l'activation β_r de chacun des N prototypes flous est d'abord calculée. Ensuite chaque règle est appliquée et combinée par l'inférence floue de type *somme-produit* pour calculer un score s_c pour chaque classe :

$$s_c = \frac{\sum_{r=1}^{N} \beta_r s_c^r}{\sum_{r=1}^{N} \beta_r}. \tag{2}$$

Cette équation montre comment les différents prototypes participent à la reconnaissance de toutes les classes : plus un prototype est activé et plus il appartient à la classe, plus il participe au score global de cette classe.

8.2 Principes de l'apprentissage incrémental

Il y a principalement deux manières d'apprendre progressivement un système de reconnaissance à base de prototypes. La première consiste à créer des nouveaux prototypes à partir des nouvelles données disponibles. La seconde consiste à ajuster (déplacer ou déformer) les prototypes existants en fonction des nouvelles données.

Bien que la création de prototypes contribue à améliorer significativement la capacité du classifieur en termes de taux de reconnaissance, la complexité du classifieur peut devenir inacceptable, en termes de temps de calcul et de mémoire, si le nombre de prototypes devient trop conséquent. Pour cela, l'utilisation de l'ajustement des prototypes existants devient indispensable afin

d'avoir un système à la fois dynamique et léger. L'ajustement des prototypes permet d'améliorer le taux de reconnaissance sans surcharger le système, mais cette technique a ses limites. En effet il lui est difficile de faire évoluer rapidement les frontières de décisions dans les cas où les nouvelles connaissances (présentées par les nouvelles données) sont très peu représentées par les prototypes existants. Dans ces cas, la décision d'ajouter des nouveaux prototypes est intuitivement plus raisonnable.

On peut donc constater que les deux techniques sont complémentaires, et le premier défi pour avoir un algorithme efficace d'apprentissage incrémental consiste à trouver la meilleure stratégie pour associer ces deux techniques.

En outre, nous supposons que les processus d'apprentissage et d'adaptation sont supervisés : chaque exemple est étiqueté correctement. Cet étiquetage est possible en demandant au scripteur de vérifier le résultat de la reconnaissance ou à l'aide d'une technique auto-supervisée comme dans [22].

Nous présentons ci-dessous les techniques de création et d'ajustement des prototypes que l'on utilise dans notre système. Nous allons ensuite présenter dans la section 9 deux stratégies d'apprentissage incrémental de systèmes à base de prototype.

8.2.1 Création de nouveaux prototypes

Ajouter un nouveau prototype dans un SIF revient à ajouter une nouvelle règle. Ce processus se compose donc de deux phases : la création du prototype puis le calcul des conclusions associées. Un prototype est créé dans l'espace des caractéristiques en choisissant un centre qui détermine la position du prototype, ainsi qu'une matrice de covariance qui représente la zone d'influence du prototype.

Dans les systèmes d'apprentissage non-incrémental, où l'on construit le système en utilisant des bases d'apprentissage avec un nombre important de données, les prototypes flous (les centres et les matrices de covariance) sont généralement appris séparément sur chaque classe en utilisant un algorithme de classification non supervisée [7]. Comme nous l'avons mentionné dans l'introduction, un des critères fondamentaux dans les systèmes d'apprentissage incrémental est de s'abstraire de l'accès aux « anciennes » données.

Ceci évite la sauvegarde systématique des données et donc allège le système, mais nécessite d'apprendre dès la première nouvelle donnée sans attendre la constitution d'un historique. Pour cela, au moment où notre système décide d'ajouter une nouvelle règle, il est obligé de créer dans l'espace des caractéristiques un prototype hyper-sphérique à partir de la seule donnée courante. Une première solution simple consiste à initialiser le centre du prototype par la donnée elle-même, et la matrice de covariance par une matrice proportionnelle à la matrice identité (on fixe donc la forme initiale du prototype).

Ces prototypes se déformeront alors, au fur et à mesure de l'apprentissage, pour avoir une forme hyper-ellipsoïdale grâce à la technique d'ajustement présentée dans la section 8.2.2. Nous présenterons, dans la section 10, une optimisation de cette phase de création de prototypes en prédisant directement la forme hyper-ellipsoïdale du prototype créé grâce à la synthèse de caractères artificiels.

8.2.2 Ajustement des prototypes existants par la méthode ADAPT

D'après l'architecture des SIF présentée dans la section 8.1, il existe plusieurs possibilités pour ajuster les SIF à de nouvelles connaissances, nous parlerons alors d'adaptation. Pour adapter les règles existantes nous pouvons modifier les prémisses et/ou les conclusions. Les prémisses étant des prototypes nous proposons deux modifications possibles : les déplacer ou les déformer pour mieux représenter la répartition des données dans l'espace des caractéristiques. Les conclusions sont modifiées pour ré-estimer la participation du prototype à chaque classe.

Nous utilisons pour cela la méthode ADAPT résultant de nos travaux précédents [18, 21]. Nous en rappelons les grands principes ci-après.

Déplacement des centres des prototypes. L'approche ADAPT s'inspire de l'algorithme LVQ (*Learning Vector Quantization*) utilisé pour l'apprentissage des cartes de Kohonen [12] : le déplacement d'un prototype consiste à modifier la position de son centre μ_r par le vecteur $\Delta\mu_r$:

$$\vec{\mu_r}(t+1) = \vec{\mu_r}(t) + \Delta\vec{\mu_r} \ . \tag{3}$$

La vitesse d'adaptation dépend d'un paramètre d'adaptation λ compris entre 0 et 1.

Contrairement à l'approche LVQ, l'approche ADAPT ne tient pas compte de l'étiquette des prototypes. Elle se base sur leur participation dans la reconnaissance de chaque classe ainsi que leur degré d'activation par l'exemple. Ainsi le déplacement du prototype doit être d'autant plus important :

- qu'il est fortement activé par l'exemple,
- qu'il participe à la reconnaissance de chaque classe,
- que l'erreur commise pour chaque classe est grande.

Ces principes intuitifs sont traduits par un facteur d'apprentissage noté δ_r défini grâce à l'interprétabilité des SIF par l'équation 4. Le déplacement $\Delta\vec{\mu}_r$ est donc définie ainsi :

$$\delta_r = \beta_r \sum_{c=1}^{C} \left((b_c - s_c)\, s_c^r \right) \ , \tag{4}$$

$$\Delta\vec{\mu}_r = \lambda \delta_r (\vec{X} - \vec{\mu_r}) \ , \tag{5}$$

avec b_c le score objectif pour s_c : 1 si c est la classe de X et 0 sinon.

Ainsi ADAPT effectue un compromis entre améliorer le score de la classe de l'exemple et ne pas augmenter le score des autres classes.

Déformation des zones d'influences des prototypes. Si le déplacement des prototypes permet de recentrer les prototypes sur les données du scripteur, la déformation de ces prototypes flous permet ensuite de mieux représenter la répartition des données décrites.

ADAPT [18, 21] utilise l'estimation itérative de la matrice de covariance comme cela a déjà été fait dans [6] pour l'EFCL (Elliptical Fuzzy Competitive Learning). Dans l'EFCL, l'apprentissage de la matrice est non supervisé et ne prend en compte que le vecteur $(\vec{X} - \vec{\mu_r})$, pondéré par un facteur d'apprentissage α :

$$Q_r^{-1} \Leftarrow \frac{Q_r^{-1}}{1-\alpha} - \frac{\alpha}{1-\alpha} \cdot \frac{(Q_r^{-1}\vec{w}) \cdot (Q_r^{-1}\vec{w})^T}{1 + \alpha(\vec{w}^T Q_r^{-1} \vec{w})} \ . \tag{6}$$

ADAPT modifie cette estimation pour prendre en compte tous les paramètres des SIF. Pour cela elle utilise le facteur d'apprentissage δ_r de l'équation 4 en remplaçant le facteur d'apprentissage α de l'équation 6 par $\alpha\delta_r$. Ainsi pour ré-estimer la matrice de covariance inverse de manière supervisée en utilisant les connaissances disponibles dans le SIF, nous obtenons l'équation :

$$Q_r^{-1} \Leftarrow \frac{Q_r^{-1}}{1-\alpha\delta_r} - \frac{\alpha\delta_r}{1-\alpha\delta_r} \cdot \frac{(Q_r^{-1}\vec{w}) \cdot (Q_r^{-1}\vec{w})^T}{1+\alpha\delta_r(\vec{w}^T Q_r^{-1}\vec{w})} \ . \tag{7}$$

Grâce à cette nouvelle estimation, la déformation, comme le déplacement, tiennent compte de l'ensemble des prototypes pour optimiser l'adaptation.

Adaptation des conclusions des règles. Pour effectuer l'adaptation des conclusions des SIF, nous utilisons la méthode classique de descente du gradient de l'erreur sur les conclusions des règles. Dans ce cas, cette approche est simple, peu coûteuse en ressources et permet une interprétation cohérente. Plus une règle est activée et plus l'erreur est grande sur le score, plus il faut modifier la conclusion.

La mise à jour des poids est donnée par l'équation 8.

$$\Delta s_c^r = n * (b_c - s_c) * \beta_r \ , \tag{8}$$

avec n le facteur d'apprentissage des poids compris entre 0 et 1. Il faut remarquer que cette mise à jour ne tient pas compte du score existant s_c^r liant la règle à la classe et encore moins de la classe sur laquelle le prototype de la règle a été appris. Ainsi, les prototypes peuvent facilement changer de classe.

Facteurs d'adaptation décroissants. Les valeurs des facteurs d'apprentissage permettent d'influencer la vitesse et la robustesse de l'adaptation. En effet les paramètres λ et α utilisés pour le déplacement et la déformation des prototypes contrôlent l'influence de chaque exemple sur l'apprentissage des paramètres du système : une valeur forte permet une adaptation rapide mais instable, une valeur faible permet une adaptation stable mais lente.

Ainsi la valeur de λ décroît de λ_{max} à λ_{min} en suivant cette équation :

$$\lambda(t) = \frac{\lambda_{max} - \lambda_{min}}{1 + t/T} + \lambda_{min} \ , \tag{9}$$

avec t le nombre de caractères entrés par l'utilisateur. Le paramètre T permet donc de régler la vitesse de décroissance de la courbe (par exemple $T = 100$ dans nos expérimentations). De la même façon le facteur de déformation α décroît de α_{max} à α_{min} en suivant la même équation.

9 Stratégies d'apprentissage incrémental

Le premier défi dans la conception d'un système d'apprentissage incrémental à base de prototypes est de répondre à ces deux questions : quand et comment créer un nouveau prototype ? Nous présentons dans cette section deux stratégies différentes pour gérer le processus de création de nouveaux prototypes dans un système d'apprentissage incrémental.

9.1 Stratégie 1 : Apprentissage incrémental en 2 phases

La première stratégie que l'on propose consiste à distinguer 2 phases différentes dans le processus d'apprentissage incrémental : la phase d'apprentissage incrémental rapide (phase 1) et la phase d'apprentissage par adaptation (phase 2).

Au début du processus d'apprentissage d'une nouvelle classe, le fait de créer de nouveaux prototypes est indispensable, parce que la classe n'est a priori pas encore modélisée par les prototypes existants. Toutefois, parallèlement à la création d'un nouveau prototype, la technique d'adaptation est toujours utile afin d'ajuster progressivement la forme et la position des prototypes dans l'espace de classification.

Rappelons qu'étant donné les contraintes de place mémoire et de temps de calcul des périphériques mobiles (PDA, smartphones) visés en application, le nombre de prototypes créés doit être raisonnable. Ainsi, en seconde phase, seule la technique d'adaptation persiste dans le processus d'apprentissage incrémental. Le passage de la phase 1 à la phase 2 a lieu après l'introduction de N données (exemples) de la classe en question. L'algorithme 1 formalise cette stratégie.

Algorithme 1 : L'algorithme d'apprentissage incrémental en 2 phases

```
pour chaque nouvel exemple e de la classe C faire
    si e est le premier exemple de la classe C alors
        créer un nouveau prototype autour de e;
        appeler l'algorithme d'adaptation;
        phase1[C] = vrai;
    sinon
        si phase1[C] alors
            créer un nouveau prototype autour de e;
            appeler l'algorithme d'adaptation;
            si nombre d'exemples de classe C ≥ N alors
                phase1[C] = faux;
        sinon
            appeler l'algorithme d'adaptation;
        fin
    fin
fin
```

9.2 Stratégie 2 : Apprentissage incrémental guidé par le rejet d'ambiguïté

Avant de présenter cette nouvelle stratégie, nous allons définir la notion de rejet d'ambiguïté qui est à la base de son principe.

9.2.1 Définition du rejet d'ambiguïté

Le but du rejet d'ambiguïté est d'évaluer la fiabilité du classifieur en détectant les formes pour lesquelles le classifieur a de fortes chances de faire une mauvaise classification, c'est à dire d'associer la forme à une mauvaise classe. Ces erreurs sont proches des frontières de décision car elles activent fortement au moins deux classes de façon équivalente. Un rejet d'ambiguïté peut être vu comme un refus de classification pour éviter une confusion.

Une technique classique pour réaliser le rejet d'ambiguïté est de définir un couloir autour des frontières de décision. Toutes les formes à l'intérieur

sont considérées comme des erreurs potentielles et sont donc rejetées. La figure 1 montre un exemple en deux dimensions de rejet d'ambiguïté pour trois classes.

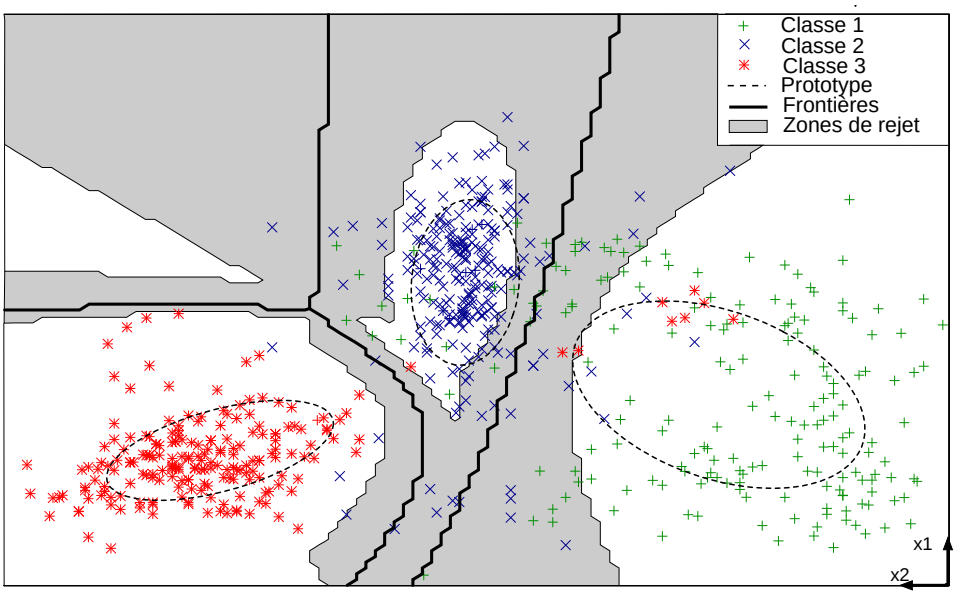

FIGURE 1 – Exemple de zones de rejet d'ambiguïté

Nous utilisons pour définir ces zones de rejet le principe des fonctions de confiance comme défini dans [19]. Ici la fonction de confiance $\psi(X)$ représente le degré d'ambiguïté lors de la classification d'un exemple X :

$$\psi(X) = (Sc_1(X) - Sc_2(X))/Sc_1(X) \tag{10}$$

où $Sc_1(X)$ est le score obtenu par la meilleure classe et $Sc_2(X)$ est le score de la second meilleure classe. Un exemple X est alors rejeté quand le degré d'ambiguïté est inférieure à un seuil défini. La valeur du seuil représente la largeur du couloir autour des frontières de décision. Ainsi pour que la classification d'un exemple soit accepté, il faut que le score de la meilleure classe soit relativement plus grand que celui de la seconde classe, ce qui correspond bien à notre objectif d'évaluation de la qualité de la classification.

9.2.2 Seconde stratégie d'apprentissage

La seconde stratégie d'apprentissage incrémental proposée repose sur la détection d'un rejet d'ambiguïté pour déclencher le processus de création de nouveaux prototypes. [1]

Le but est de limiter la création de nouveaux prototypes pour une classe donnée en s'appuyant sur le rejet d'ambiguïté pour éviter de créer des prototypes dans les endroits où il n'y a pas de risque de confusion. Cela élimine l'ajout « inutile » de prototypes pour une classe donnée dans des zones de l'espace où cette classe est déjà dominante.

Le principe de cette stratégie repose toujours sur l'idée de favoriser la création de prototypes au début, puis de diminuer progressivement le nombre des créations en comptant sur la technique d'adaptation pour continuer l'apprentissage. L'ajustement du nombre de prototypes créés est assuré par la modification de la valeur du seuil de rejet d'ambiguïté pour une classe donnée. On utilise un seuil de moins en moins strict à chaque fois que l'on ajoute un prototype pour cette classe.

L'avantage de cette stratégie est que pour les classes dites « faciles », la création de nouveaux prototypes sera rapidement restreinte étant donné l'absence d'ambiguïtés. En revanche, le système peut continuer à ajouter des prototypes pour modéliser les classes dites « difficiles » tant que des conflits sont toujours déclenchés. L'algorithme 2 décrit cette stratégie.

9.3 Apprentissage des conclusions

Pour chaque nouvelle règle, les conclusions sont initialisées par 1 pour la classe qui correspond à cette règle et 0 pour toutes les autres classes. Ensuite, ces conclusions sont incrémentalement apprises par une descente de gradient sur une mémoire partielle qui contient les N derniers exemples traités par le système. La taille de cette mémoire partielle est limitée dans nos expérimentations à 30.

[1] Le rejet est juste utilisé pour guider l'apprentissage ; il n'y pas d'option de rejet sur la réponse finale du système.

Algorithme 2 : L'algorithme d'apprentissage incrémental guidé par le rejet d'ambiguïté

```
pour chaque nouvel exemple e de la classe C faire
    si e est le premier exemple de la classe C alors
        créer un nouveau prototype autour de e;
        appeler l'algorithme d'adaptation;
        seuil_rejet[C] = valeur initiale prédéfinie ;
    sinon
        appeler l'algorithme d'adaptation;
        calculer le degré d'ambiguïté;
        si rejet d'ambiguïté alors
            créer un nouveau prototype autour de e;
            diminuer la valeur de seuil_rejet[C];
        fin
    fin
fin
```

10 Accélération de l'apprentissage par la synthèse

Afin d'obtenir un apprentissage incrémental rapide et de surmonter le manque de données disponibles au début du processus d'apprentissage, nous choisissons de synthétiser des données artificielles. La solution idéale serait de pouvoir déduire directement les matrices de covariance associées aux prototypes en évitant la synthèse, mais ce problème est très complexe dans l'espace des caractéristiques. Or, notre expérience dans le domaine de l'écriture manuscrite nous permet d'explorer de façon robuste les variations potentielles de la formes des caractères dans l'espace des coordonnées. Nous allons détailler dans la section 10.1 les techniques de synthèse utilisées, puis nous présentons dans la section 10.2 l'optimisation du processus de création de prototypes grâce à la synthèse de données artificielles.

10.1 Synthèse des caractères manuscrits

10.1.1 État de l'art

La synthèse de caractères artificiels pour augmenter la quantité de données disponibles lors d'un apprentissage de classifieur est quelque chose d'assez souvent utilisé dans la reconnaisse de caractères en hors ligne ([3] avec un k-PPV, [27] avec un réseaux de neurones, [9, 28] avec des Modèles de Markov Cachés...). Dans notre cas, il s'agit de générer plusieurs variantes d'un même caractère à partir d'un seul exemple de ce caractère en-ligne. Pour cela nous appliquons, sur l'exemple disponible, une ou plusieurs des déformations qui peuvent s'inspirer des déformations utilisées sur les caractères hors-ligne ou qui seront spécifiques à la nature en-ligne du signal. Dans tous les cas, à chaque nouvelle génération il faut choisir des paramètres de déformation différents pour obtenir une certaine variabilité dans les données générées. Mais il ne faut pas non plus utiliser des paramètres engendrant des déformations trop importantes pour ne pas risquer de synthétiser un caractère qui ne ressemble plus du tout à celui d'origine. En effet, dans notre contexte il s'agit de conserver le style du scripteur et non de générer un nouveau style. Pour chaque paramètre, une borne minimale et une borne maximale sont fixées pour limiter les déformations générées. Ces bornes sont fixées empiriquement en visualisant pour chacune d'elles les déformations obtenues pour plusieurs classes de caractères. Une fois fixées, les bornes restent les mêmes tout au long des expérimentions.

Dans [20], nous avons présenté deux stratégies pour générer des caractères manuscrits. La première utilise les distorsions classiques de l'image (de type hors-ligne). Les déformations appliquées peuvent être assez simples comme l'inclinaison, l'étirement et la variation de l'épaisseur du trait dans [3] ou plus compliquées comme la distorsion élastique (pour créer des « vagues » sur l'image) utilisée dans [16, 27]. La seconde, fondée sur la particularité de l'écriture manuscrite en-ligne, applique des distorsions en-ligne sur le caractère : modification de la vitesse et modification de la courbure.

10.1.2 Déformations utilisées

Nous utilisons dans ces travaux quatre déformations qui correspondent à nos besoins de synthétiser de nouveaux caractères tout en conservant le style du scripteur. Les deux premières déformations sont des déformations de l'image du caractère [2] :

- **l'étirement :** Une image peut être étirée suivant l'axe x ou l'axe y. L'opération d'étirement dépend donc de deux paramètres α_x et α_y respectivement le coefficient de déformation suivant l'axe x et le coefficient de déformation suivant l'axe y. Pour un tracé en ligne, les coordonnées des points sont modifiées suivant les équations 11 et 12, $x'_{(t)}$ et $y'_{(t)}$ étant les coordonnées du tracé déformé :

$$x'_{(t)} = \alpha_x x_{(t)}, \tag{11}$$

$$y'_{(t)} = \alpha_y y_{(t)}. \tag{12}$$

- **l'inclinaison :** L'inclinaison d'une image est différente de la rotation d'une image. En effet cette déformation se rapproche plus de l'écriture « penchée ». L'inclinaison dépend d'un paramètre α_i. Comme le montrent les équations 13 et 14, la coordonnée en y ne change pas mais la variation de $x_{(t)}$ dépend de la valeur de $y_{(t)}$:

$$x'_{(t)} = x_{(t)} + \alpha_i * y_{(t)}, \tag{13}$$

$$y'_{(t)} = y_{(t)}. \tag{14}$$

Les déformations de type en-ligne utilisent des informations sur le tracé à déformer qui ne sont pas disponibles dans l'image. En effet, le signal en-ligne intègre des informations sur la dynamique (la cinétique) de l'écriture. L'ordre des points du tracé et leur cinétique peuvent être exploités dans les stratégies de déformation. Nous utilisons les deux déformations suivantes :

- **la modification de la vitesse :** L'objectif de cette déformation est de déplacer certaines parties du caractère les unes par rapport aux autres. Pour cela nous proposons de modifier la longueur des traits verticaux

[2] on suppose que $min_t(x(t)) = 0$ et $min_t(y(t)) = 0$.

ou horizontaux sans modifier les parties orientées différemment. Nous modifions la taille des vecteurs $\vec{V}_{(t-1)} = \left(x_{(t)} - x_{(t-1)}, y_{(t)} - y_{(t-1)}\right)$ suivant leur direction. Ainsi les vecteurs dont les directions sont proches des axes principaux pourront augmenter ou diminuer selon un paramètre α_v. Les vecteurs seront considérés comme proches d'un axe principal s'ils forment un angle inférieur à $\frac{\pi}{8}$ avec l'un des axes. Les équations 15 à 17 donnent les coordonnées des points du tracé après cette déformation :

$$x'_{(t)} = x'_{(t-1)} + \beta * (x_{(t)} - x_{(t-1)}), \tag{15}$$

$$y'_{(t)} = y'_{(t-1)} + \beta * (y_{(t)} - y_{(t-1)}) \tag{16}$$

$$\beta = \begin{array}{ll} 1 & \text{si } \arg(\vec{V}_{(t-1)}) \left[\frac{\pi}{2}\right] \in \left[\frac{\pi}{8}, \frac{3\pi}{8}\right] \\ \alpha_v & \text{sinon} \end{array} \tag{17}$$

- **la modification de la courbure :** Nous proposons ici de déformer les boucles et les liaisons entre deux parties rectilignes, qui sont des éléments structurants de l'écriture. Les boucles pourront s'ouvrir et se fermer et les liaisons pourront modifier légèrement le positionnement relatif des différentes parties rectilignes. Pour cela nous proposons une déformation qui agit sur les courbes du caractère et non sur les parties rectilignes. La modification de la courbure du tracé se fait en fonction d'un paramètre α_c. Pour ne pas complètement modifier le caractère et conserver sa structure, il y a certaines parties du tracé qu'il ne faut pas modifier, par exemple les lignes droites et les points de rebroussement du tracé. Ainsi la déformation sera maximale pour des angles proches de $\frac{\pi}{2}$ et nulle pour les angles 0 et π. Les équations 18 à 20 donnent la position des points $p'_{(t)}$ après déformation :

$$\hat{\theta}_{(t-1)} = (p_{(t-2)}, \widehat{p_{(t-1)}, p_{(t)}}) \in]-\pi, \pi], \tag{18}$$

$$p'_{(t)} = p^* \quad \text{tel que} \quad (p'_{(t-2)}, \widehat{p'_{(t-1)}, p^*}) = \hat{\theta}_{(t-1)} - \beta, \tag{19}$$

avec :

$$\beta = \alpha_c * 4 * \frac{\left|\hat{\theta}_{(t-1)}\right|}{\pi} * (1 - \frac{\left|\hat{\theta}_{(t-1)}\right|}{\pi}). \tag{20}$$

10.1.3 Exemples de caractères synthétiques

Les quatre déformations que nous proposons pour la synthèse de caractères sont complémentaires, c'est-à-dire qu'ils ne génèrent pas des déformations identiques. Comme notre but est d'obtenir le plus possible de variabilité autour du style du scripteur, nous appliquons pour chaque caractère synthétisé soit l'inclinaison, soit la modification de la courbure, soit la modification de la vitesse qui sont trois déformations importantes du caractère puis systématiquement un étirement qui est une déformations avec moins d'influence. Les paramètres des déformations (α_x, α_y, α_i, α_v et α_c) sont chacun tirés aléatoirement dans un intervalle de valeurs possibles. Cet intervalle fixé grâce à nos connaissances sur l'écriture manuscrite, permet de ne pas générer des caractères trop déformés et donc de conserver le style du scripteur. La figure 2 montre des exemples de caractères manuscrits générés par chaque déformation.

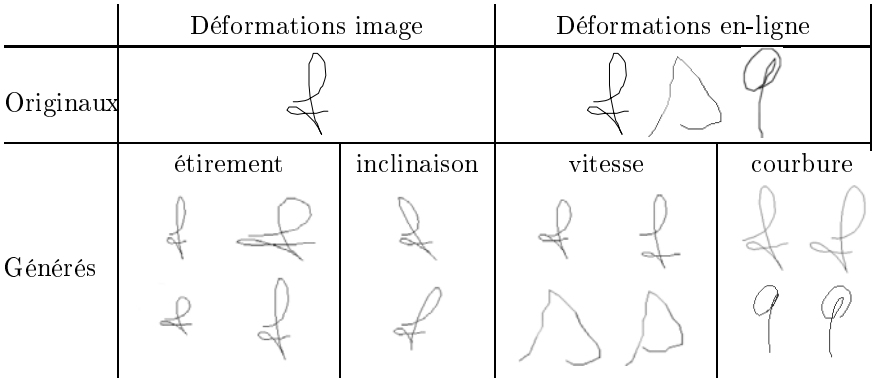

FIGURE 2 – Exemples de caractères manuscrits générés.

Nous pouvons constater que chaque génération permet des déformations des caractères d'origine qui ne sont pas atteignables par les autres stratégies. Ces stratégies permettent bien de créer des caractères différents des originaux en conservant le style du scripteur.

10.2 Création de prototypes à partir de données synthétisées

En raison du manque de connaissance au début de l'apprentissage, la synthèse de caractères manuscrits permet d'améliorer la qualité des prototypes créés. Nous utilisons donc la synthèse, dans l'espace des formes, pour créer

de nouveaux prototypes représentatifs de l'écriture naturelle du scripteur courant.

Grâce à la synthèse, un nouveau prototype est créé en calculant le centre et la matrice de covariance à partir des caractères artificiels générés, ce qui produit directement des prototypes hyper-ellipsoïdaux au lieu des prototypes hyper-sphériques comme dans la section 8.2.1.

11 Expérimentations

Ces expérimentations ont pour but d'abord de comparer les deux stratégies d'apprentissage incrémental présentées dans la section 9, et ensuite de montrer le rôle de la synthèse de caractères artificiels dans le processus d'apprentissage. Nous présentons notre protocole expérimental dans la section 11.1, pour ensuite synthétiser les résultats obtenus dans les sections 11.2 et 11.3.

11.1 Protocole expérimental

Les expérimentations portent sur la reconnaissance des 26 lettres latines minuscules isolées en-ligne. Les bases de caractères mono-scripteur ont été écrites sur un PDA par 18 scripteurs. Chaque scripteur a saisi 40 fois chaque caractère i.e. 1040 caractères par scripteur. Dans ces expériences, les caractères sont saisis dans un ordre aléatoire.

Pour estimer les performances de l'apprentissage incrémental sur chaque scripteur, nous procédons à un test mono-scripteur en utilisant le principe de la validation croisée stratifiée en quatre parties. Trois quarts de la base du scripteur (780 lettres) sont utilisés pour l'apprentissage incrémental et un quart (260 lettres) est utilisée pour évaluer le taux de reconnaissance du système tout au long du processus (base de test). Les résultats que nous présentons correspondent à la moyenne du taux de reconnaissance sur les 18 scripteurs. Dans ces expérimentations, les caractères sont décrits par 21 caractéristiques. Un exemple de chaque classe est présenté au système à chaque cycle d'apprentissage.

Dans la stratégie 1, la phase d'apprentissage incrémental rapide pour une classe donnée se termine après avoir présenté 10 exemples de cette classe. Le

système passe ensuite à la seconde phase d'apprentissage par adaptation. Pour la stratégie 2, la politique de diminution du seuil d'ambiguïté, au fur et à mesure de l'apprentissage, est illustré par la figure 3. Cette politique permet de favoriser la création de prototypes au début, puis de diminuer progressivement le nombre des créations en comptant sur la technique d'adaptation pour continuer l'apprentissage.

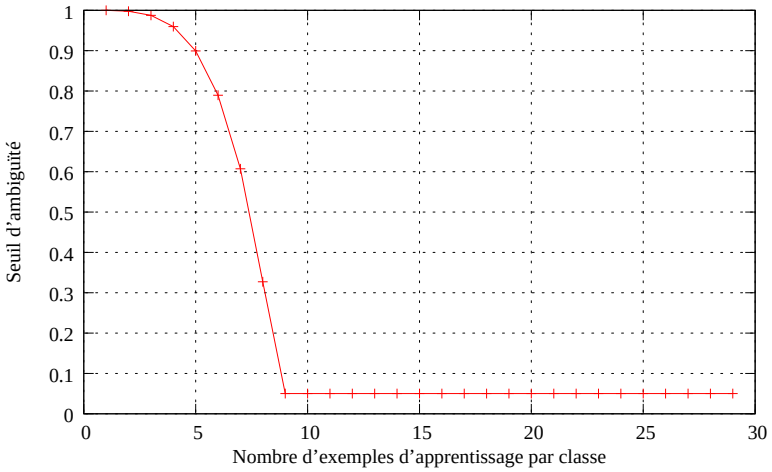

FIGURE 3 – Politique de diminution du seuil d'ambiguïté

11.2 Évaluation des stratégies d'apprentissage incrémental

Nous comparons dans cette première expérimentation la performance des deux stratégies d'apprentissage incrémental en termes de complexité du classifieur produit d'une part, et de qualité du classifieur d'autre part. La complexité du classifieur est relative à l'espace mémoire utilisé et au temps de calcul nécessaire pour effectuer la reconnaissance d'un caractère. Dans notre système, cet espace mémoire et ce temps de calcul dépendent linéairement du nombre de prototypes créés et utilisés par le système. La figure 4 montre le nombre moyen de prototypes utilisés dans le SIF pour l'ensemble des classes lors du processus d'apprentissage incrémental. Nous avons ajouté dans cette expérimentation ce que nous avons appelé la « Stratégie 0 » dans laquelle le système crée systématiquement un nouveau prototype pour chaque nouvel exemple rencontré. Cette stratégie n'est pas applicable en pratique mais elle met en évidence que le fait de créer en permanence de nouveaux prototypes

n'améliore pas forcement la qualité du classifieur.

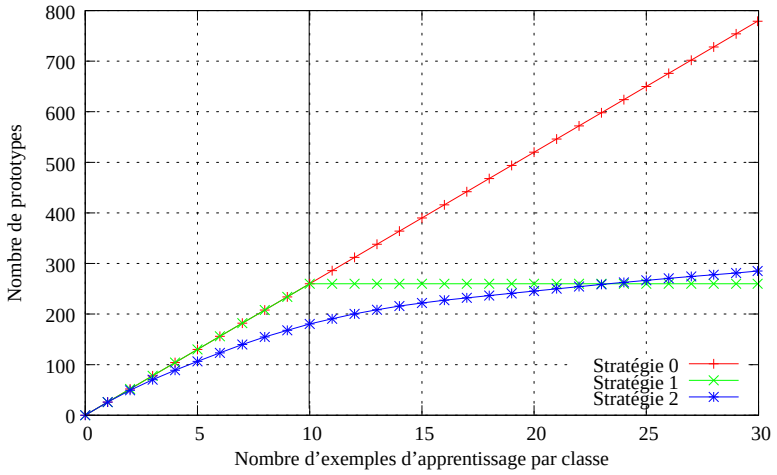

FIGURE 4 – Évolution du nombre de prototypes lors du processus d'apprentissage

La qualité du classifieur est évaluée par le taux de reconnaissance obtenu avec la base de test. La figure 5 montre l'évolution du taux moyen de reconnaissance pour les 18 scripteurs.

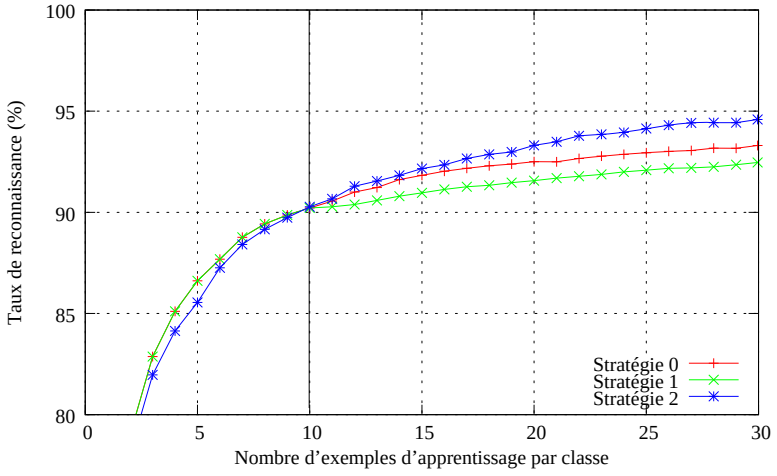

FIGURE 5 – Évolution du taux de reconnaissance lors du processus d'apprentissage

A travers ces deux figures, nous constatons que la stratégie 2 produit un classifieur de qualité égale ou supérieure à celui obtenu avec la stratégie 1,

tout en créant moins de prototypes.

Si l'on fait abstraction de l'axe du temps représenté par le nombre d'exemples d'apprentissage par classe dans les figures 4 et 5, on met en relief le rapport qualité/complexité du classifieur obtenu avec chaque stratégie (cf. figure 6). On constate que la stratégie 2 permet d'atteindre un meilleur taux de reconnaissance pour un même nombre de prototypes.

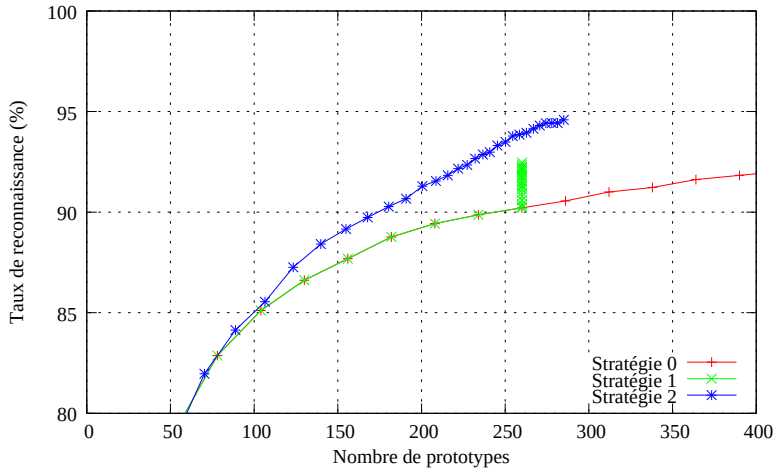

FIGURE 6 – Rapport qualité/complexité du classifieur obtenu avec chaque stratégie

11.3 Évaluation du rôle de la synthèse

Le but de cette expérimentation est de montrer l'impact de la synthèse de caractères artificiels sur la qualité et la complexité du classifieur dans notre algorithme d'apprentissage incrémental.

L'impact de la synthèse sur la qualité du classifieur est illustré par la figure 7. Nous constatons qu'en utilisant la stratégie 2 avec la synthèse de caractères, un taux de reconnaissance d'environ 90% est réalisé avec seulement 5 exemples d'apprentissage. Ce taux augment progressivement pour atteindre 94% pour 10 exemples et environ 97% après 30 exemples. L'utilisation de la synthèse fait diminuer l'erreur de reconnaissance de 40% en moyenne.

La figure 8 montre l'impact de la synthèse sur la complexité du classifieur. Puisque la création de prototypes dans la stratégie 1 est systématique dans

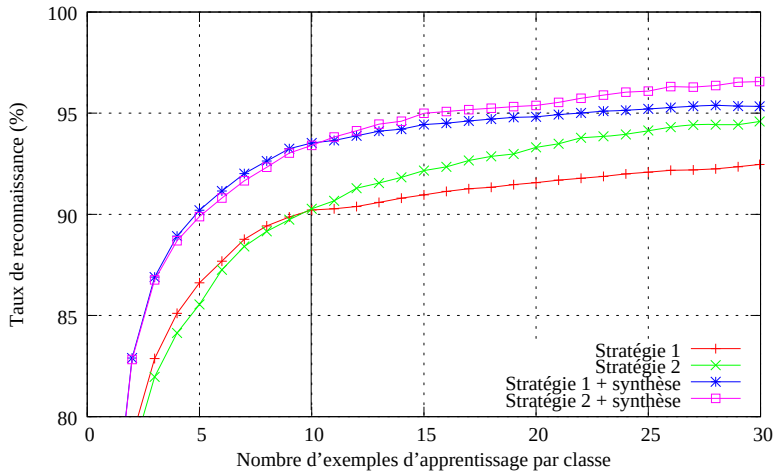

FIGURE 7 – Impact de la synthèse sur l'évolution du taux de reconnaissance

la phase 1 et inhibée dans la phase 2, la synthèse n'a pas d'impact sur cette représentation pour la stratégie 1. Contrairement à la stratégie 2 dans laquelle plus le système est performant et moins on crée de prototypes. Cela est confirmé dans la figure 8 où nous remarquons que l'utilisation de la synthèse dans la stratégie 2 permet de diminuer le nombre de prototypes dans le système.

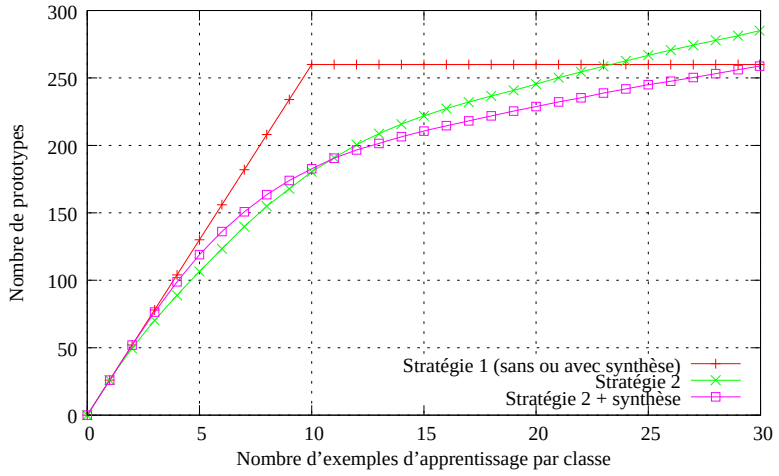

FIGURE 8 – Impact de la synthèse sur l'évolution du nombre de prototypes de SIF

La figure 9 confirme l'amélioration du rapport qualité/complexité du clas-

sifieur grâce à l'utilisation de la synthèse.

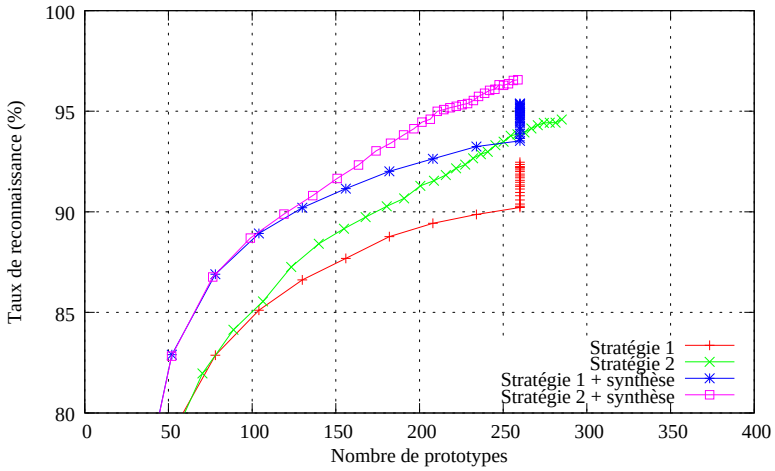

FIGURE 9 – Rapport qualité/complexité du classifieur avec la synthèse

12 Conclusion et travaux futurs

Dans le contexte de la reconnaissance de l'écriture manuscrite en-ligne, nous avons présenté un nouvel algorithme d'apprentissage incrémental basé sur un système d'inférence floue. Grâce à cet algorithme, le système de reconnaissance est capable d'apprendre de nouvelles formes à partir de très peu de données. Il peut de plus s'adapter et s'améliorer pour chaque nouvelle donnée disponible. Nous avons intégré des techniques de synthèse de caractères manuscrits pour accélérer l'apprentissage et ainsi améliorer le taux de reconnaissance.

Les travaux futurs à court terme consisteront à diminuer le nombre de prototypes « inutiles » dans le système soit en supprimant les prototypes peu activés, où en fusionnant les prototypes redondants. A plus long terme, nous envisageons d'explorer d'autres approches pour synthétiser des caractères manuscrits en s'inspirant notamment des techniques proposées par Plamondon[23] sur le modèle Sigma-lognormal.

13 Remerciement

Les auteurs souhaitent remercier Guy Lorette, Professeur à l'Université de Rennes I, pour ses précieux conseils.

Références

[1] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Mach. Learn.*, 6(1) :37–66, 1991.

[2] Bernadette Bouchon-Meunier and Christophe Marsala, editors. *Logique Floue, Principes, Aide à la Décision.* Hermès-Lavoisier, 2003.

[3] Javier Cano, Juan-Carlos Pérez-Cortes, Joaquim Arlandis, and Rafael Llobet. Training set expansion in handwritten character recognition. In *Proceedings of the 9th International Workshop on Structural and Syntactic Pattern Recognition (SSPR) and 4th Statistical Pattern Recognition (SPR)*, pages 548–556, 2002.

[4] G. Carpenter, S. Grossberg, N. Markuzon, J. Reynolds, and D. Rosen. Fuzzy artmap : A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3, 1992.

[5] Gail A. Carpenter and Stephen Grossberg. The art of adaptive pattern recognition by a self-organizing neural network. *Computer*, 21(3) :77–88, 1988.

[6] S. De Backer and P. Scheunders. Texture segmentation by frequency-sensitive elliptical competitive learning. *Image and Vision Computing*, 19(9–10) :639–648, 2001.

[7] G. Lorette E. Anquetil. Automatic generation of hierarchical fuzzy classification systems based on explicit fuzzy rules deduced from possibilistic clustering : Application to on-line handwritten character recognition. In *in Proceedings of the international conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, (IPMU'96)*, 1996.

[8] Phayung Meesad Gary G. Yen. An effective neuro-fuzzy paradigm for machinery condition health monitoring. *IEEE Transactions on Systems, Man, and Cybernetics*, 31-4 :523 – 536, 2001.

[9] Muriel Helmer and Horst Bunke. Generation and use of synthetic training data in cursive handwriting recognition. In *Proceedings of 1st Iberian Conference Pattern Recognition and Image Analysis (IbPRIA)*, pages 336–345, 2003.

[10] J.-S.R. Jang. Anfis : adaptive-network-based fuzzy inference system. *Systems, Man and Cybernetics, IEEE Transactions on*, 23 :665–685, 1993.

[11] Joseph J. LaViola Jr. and Robert C. Zeleznik. A practical approach for writer-dependent symbol recognition using a writer-independent symbol recognizer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11) :1917–1926, 2007.

[12] Teuvo Kohonen. The self-organizing map. *Proceedings of IEEE*, 78(9) :1464–1480, 1990.

[13] Nicholas Littlestone. Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. In *COLT '91 : Proceedings of the fourth annual workshop on Computational learning theory*, pages 147–156, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[14] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2) :212–261, 1994.

[15] Edwin Lughofer. Flexfis : A robust incremental learning approach for evolving takagi-sugeno fuzzy models. *IEEE T. Fuzzy Systems*, 16(6) :1393–1410, 2008.

[16] Farès Menasri, Nicole Vincent, and Emmanuel Augustin. Reconnaissance de chiffres farsi isolés par réseau de neurones à convolutions. In *Actes du Colloque International Francophone sur l'Ecrit et le Document (CIFED'08)*, pages 127–132, 2008.

[17] Fernanda Li Minku, Hirotaka Inoue, and Xin Yao. Negative correlation in incremental learning. *Natural Computing : an international journal*, 8(2) :289–320, 2009.

[18] H Mouchere, E Anquetil, and N Ragot. On-line writer adaptation for handwriting recognition using fuzzy inference systems. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 21(1) :99–116, 2007.

[19] Harold Mouchere and Eric Anqueti. A unified strategy to deal with different natures of reject. In *18th ICPR*, 2006.

[20] Harold Mouchere and Eric Anquetil. Synthèse de caractères manuscrits en-ligne pour la reconnaissance de l'écriture. In *Actes du Colloque International Francophone sur l'Ecrit et le Document (CIFED'06)*, pages 187–192, 2006.

[21] Harold Mouchère. *Étude des mécanismes d'adaptation et de rejet pour l'optimisation de classifieurs : Application à la reconnaissance de l'écriture manuscrite en-ligne*. PhD thesis, Institut National des Sciences Appliquées de Rennes (INSA), 2007.

[22] Loïc Oudot, Lionel Prevost, Alvaro Moises, and Maurice Milgram. Self-supervised writer adaptation using perceptive concepts : Application to on-line text recognition. In *17th ICPR*, volume 2, pages 598–601, 2004.

[23] Réjean Plamondon and Moussa Djioua. A multi-level representation paradigm for handwriting stroke generation. *Human Movement Science*, 25 :586–607, October 2006.

[24] Robi Polikar, Lalita Udpa, Satish Udpa, and Vasant Honavar. Learn++ : An incremental learning algorithm for supervised neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 31 :497–508, 2001.

[25] R Reinke and R Michalski. Incremental learning of concept descriptions : A method and experimental results. *Machine Intelligence*, 11 :263–288, 1988.

[26] Javad Sadri, Ching Y. Suen, and Tien D. Bui. A new clustering method for improving plasticity and stability in handwritten character recognition systems. *Pattern Recognition, International Conference on*, 2 :1130–1133, 2006.

[27] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best practice for convolutional neural network applied to visual analysis. In *Proceedings of the 7th International Conference on Document Analysis and Recognition (ICDAR)*, 2003.

[28] Tamás Varga and Horst Bunke. Generation of synthetic data for an hmm-based handwriting recognition system. In *Proceedings of 7th International Conference on Document Analysis and Recognition (ICDAR)*, pages 618–622, 2003.

Abdullah Almaksour est ingénieur diplômé de l'université d'Alep. Il prépare actuellement une thèse en informatique de l'Institut National des Sciences Appliquées (INSA) de Rennes, au sein de l'équipe IMADOC à l'IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires). Ses thèmes de recherche concernent les systèmes d'apprentissage incrémental, les systèmes d'inférence floue, la reconnaissance de gestes manuscrits et la synthèse de l'écriture manuscrite.

Harold Mouchère a obtenu un doctorat en informatique à l'Institut National des Sciences Appliquées (INSA) de Rennes, France, en 2007.

Il s'intéresse à la reconnaissance des formes et plus particulièrement à l'optimisation de classifieurs flous par adaptation ou par modélisation des connaissances grâce à des options de rejet. Après quatre années dans l'équipe de recherche IMADOC de l'IRISA (Institut de Recherche en Informatique et Systèmes Aléatoires), il intègre l'équipe de recherche IVC de l'IRCCyN (Institut de Recherche en Communication et Cybernétique de Nantes) en 2008 en tant que Maîtres de Conférence à l'Université de Nantes. Il travaille actuellement sur la reconnaissance d'expressions mathématiques et sur l'identification de scripteur.

Eric Anquetil est ingénieur et docteur en informatique. Depuis 1997, il est maître de conférences à l'INSA de Rennes. Il est aujourd'hui habilité à diriger des recherches qu'il effectue au sein de l'équipe IMADOC au laboratoire de l'IRISA. Après avoir longtemps travaillé sur la reconnaissance d'écriture manuscrite en-ligne, ses recherches s'orientent maintenant vers la reconnaissance " à la volée " de documents manuscrits complexes : schéma électrique, plan d'architecture, partition musicale... Il est notamment spécialiste des approches de reconnaissance de formes hybrides " statistiques/structurelles " à base de logique floue.