Université Laval | GLO 7001

Conception de Systèmes Intelligents

Rapport de Projet

Système de recommandation de flashcards gamifié

Réalisé par
Anas Neumann, NI° 111-189-477

Session d'automne 2019

Sommaire

I. Introduction

I.1. Contexte général

Ce travail s'inscrit dans le cadre d'un partenariat entre l'Université Laval (et plus particulièrement le module *GLO 7001 - Conception de systèmes intelligents)* et le projet à but non lucratif *Wikimedica*[1]. Ce partenariat à pour but d'apporter une dimension professionnelle aux projets des étudiants qui répondront alors à un besoin réel et permet également de contribuer à un projet qui s'inscrit parfaitement dans les valeurs de l'Université Laval. En effet, *Wikimédica* est une plateforme libre et gratuite de partage d'informations relatives à la santé et aux soins.

Figure 1. Logo de Wikimedica

Dans ce contexte, *Wikimedica* a proposé 3 axes d'amélioration qui leur semblent utiles et, ensemble, nous avons tracé les prémisses de 3 projets possibles rejoignant à la fois les objectifs de *Wikimedica* et l'ajout d'Intelligence Artificielle :

1. Les FlashCards : La plateforme *Wikimedica* comprend aujourd'hui un système de *FashCards*[2] relativement simple qui se présente sous la forme d'une liste statique de questions-réponses. A cela, il serait possible d'ajouter de l'apprentissage personnalisé (recommandation selon l'historique) ou de l'aide à l'édition (génération de cartes, détection des doublons ou erreurs...).
2. Les algorithmes cliniques : De la même manière, les algorithmes cliniques sont aujourd'hui une simple image statique récapitulant les étapes à suivre par un médecin pour la détection ou le soin d'une maladie en particulier. On pourrait ajouter à cela un outil interactif de création ou modification des algorithmes, de la génération automatique ou de la détection de liens entre les différents algorithmes.
3. Les ontologies : *Wikimedica* possède d'ores et déjà une base d'ontologies dans le but d'apporter de la sémantique aux données médicales présentes sur la plateforme. Cependant ils souhaiteraient ajouter un outil permettant d'alimenter la base actuelle à partir d'autres bases ouvertes en ligne (notamment anglophones).

I.2. Présentation du projet

Ce travail s'inscrit alors dans le cadre du projet n°1 (les *FlashCards*) et plus particulièrement dans l'axe autour de l'apprentissage personnalisé. Nous allons donc concevoir et développer un outil permettant pour chaque utilisateur d'obtenir les cartes qui lui seront le plus utiles (en fonction de son historique échecs-réussites, des cartes et thèmes qu'il n'a pas encore abordé ou encore de son niveau et de la difficulté des cartes). A cela nous allons ajouter une dimension "*gamification*" (parfois

[1] Voir le lien de la plateforme : *https://wikimedi.ca/wiki/Accueil*
[2] Carte d'apprentissage comportant une question à l'avant et une réponse cachée à l'arrière

appelée en français "*ludification*") afin de rendre le système plus attractif pour l'apprenant.

Comme on peut le voir sur la Figure 2, nous souhaitons, pour deux raisons très simples, que le système soit dissocié de la plateforme actuelle. Premièrement, il s'agit d'une raison technologique : la plateforme *Wikimedica* est développée entièrement avec le *CMS MediaWiki*[3] qui ne permet pas l'intégration de certaines technologies dont on aura besoin ainsi que de certaines données[4]. La seconde raison est plutôt pratique : les apprenants étant majoritairement des étudiants, il serait plus aisé que le système de FlashCards soit une tierce application directement accessible sur web et mobile.

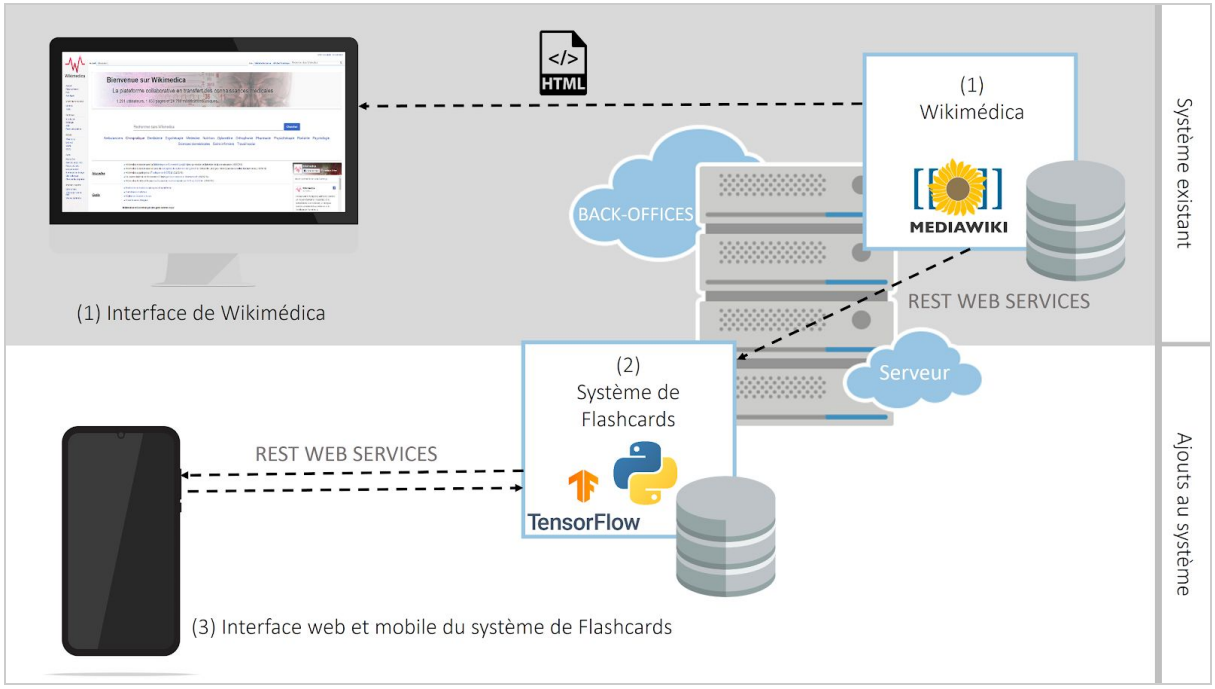

Figure 2. Architecture générale du projet

Ce projet se concentre sur l'application **(2)** de la Figure 2 (dite "*back office*"), c'est à dire le système de recommandation sur serveur. Ce système est lié à la fois à **(1)** la plateforme existante, et à **(3)** l'interface sous forme d'application mobile. L'interface du système en **(3)**, ne faisant pas réellement partie du système "*intelligent*", sera évoquée mais pas développée.

L'aspect "*intelligent*" du système **(2)** repose, quant à lui, sur l'utilisation d'algorithmes du "*Machine Learning*" pour la classification des cartes par difficulté et thème. En effet, cela pourra être réalisé via des algorithmes tels que *LDA* ou des *Réseaux de Neurones Artificiels (RNA)* que nous allons détailler par la suite.

I.3. Approche méthodologique

La méthodologie employée pour la réalisation de ce projet est assez simple et repose sur 3 principales étapes :

1. **Revue de la littérature scientifique et technologique** : afin d'en apprendre plus sur

[3] Voir le site web officiel de MediaWiki *: https://www.mediawiki.org/wiki/MediaWiki*
[4] Voir le modèle de données (méta-modèle) de MediaWiki :
https://upload.wikimedia.org/wikipedia/commons/9/94/MediaWiki_1.28.0_database_schema.svg

les différentes méthodes existantes pouvant répondre à notre besoin, les choix conceptuels à faire et les conseils à suivre.

2. **Choix des outils** : dans notre cas, il s'agit principalement TensorFlow [61] et des réseaux de neurones pour la classification ou encore LDA pour la découvertes des thèmes. Pour cela nous utilisons divers critères de sélection : les impacts des librairies choisies sur les plateformes réputées (notes et utilisations sur GitHub ou encore support sur les Cloud tels que AWS) ou encore le nombre de citations des articles (l'article fondateur de LDA [11] est cité, à ce jour, plus de 28 500 fois).
3. **Conception du projet** : pour cela nous allons utiliser *UML*[5] qui reste le standard pour la modélisation *Orienté Objets*.

Ce document suivra donc cette même logique. Ainsi, la partie II sera une revue des principaux concepts utilisés, à savoir : un système intelligent, les RNA, LDA et la gamification. Finalement, la partie III détaillera la conception et de l'implémentation (en *Python*) du projet.

II. Définitions et présentation des concepts

Comme nous l'avons précisé précédemment, cette seconde partie présente les concepts clés sur lesquels est fondé notre système. Cette présentation se base sur une revue de la littérature scientifique ainsi que sur des articles web provenant de sites spécialisés ou encore de *Wikipédia*[6]. Nous commencerons par rappeler ce qu'est l'*Intelligence Artificielle* et les *systèmes intelligents*. Ensuite, nous introduiront l'apprentissage machine et les deux concepts liés que nous souhaitons utiliser : les *RNA* et *LDA*. Finalement, nous allons présenter la gamification et les raisons qui nous ont motivés à vouloir intégrer cette dimension dans le projet.

II.1. Les systèmes intelligents

Bien que le terme soit apparu globalement en 1956 durant le workshop "*Dartmouth Summer Research Project on IA*" organisé notamment par John McCarthy, l'idée d'une machine capable d'intelligence revient à Alain Turing et son "Test de Turing" (1950). Cependant certains scientifiques/historiens considèrent même que la genèse de l'idée est bien antérieure aux ordinateurs et daterait de -800 avant Jésus Christ. avec des références mythologiques à des créatures (de fabrication humaine) capables d'intelligence [63, 64 ,68]. Le qualificatif "*Intelligence Artificielle*" est valable pour tous mécanismes non biologiques et particulièrement informatiques/électroniques, dits "*artificiels*", capables de simuler l'intelligence humaine et particulièrement ses fonctions cognitives (perception, mémoire, décision, mouvement ...). En effet les capacités de calcul et de logique de l'homme sont depuis bien longtemps réalisées par des machines, plus rapidement et avec plus de précision. Cependant la définition d'intelligence ne fait pas unanimité et peut globalement être divisée entre celle qui stipule que l'intelligence est la simulation du comportement humain et celle qui préfère la "rationalité", permettant ainsi d'éviter certains problèmes posés par les limites humaines [63, 64, 65, 66, 67]. Russel et Norvig ont donc conçu une classification, Figure 3, des systèmes intelligents selon leur positionnement :

[5] *"Unified Modeling Language"* est un langage de conception Orienté Objet.
[6] Encyclopédie en ligne (ouverte en lecture et écriture), qui a l'avantage d'introduire de nombreux concepts avec beaucoup de simplicité.

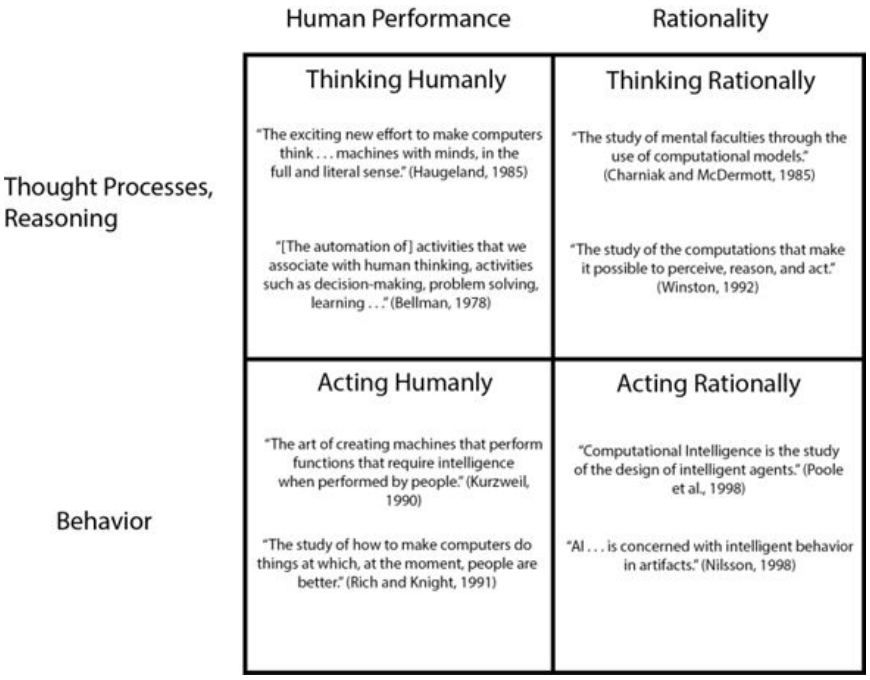

Figure 3. Modèle de Russel & Norvig

L'IA permet notamment de "*résoudre*", à l'aide d'heuristiques (méta-heuristiques ou encore algorithmes évolutionnistes) ou autres techniques (réseau de neurones, réseaux bayésiens) divers types de problèmes NP-Difficiles non-solvables en utilisant des algorithmes classiques ou les mathématiques. Ces problèmes sont extrêmement variés : classification (réseaux de neurones), analyse de documents multimédia (traitement du langage naturel, image, son), capacité de prévision et recommandation (algorithmes sur les graphes), adaptation à l'imprévu (simulation)

Ainsi, les "*système intelligents*", ou systèmes utilisant l'IA, sont tous aussi variés dans leurs utilités : on peut citer les systèmes de traduction, la reconnaissance d'écriture manuscrite, la direction automatique de véhicules, La prévision d'évènements ou de l'évolution de la mode à travers les comportements sociaux, les utilisations diverses de la robotique (médecine à distance, service dans les restaurants ...), le jeu vidéo contre l'ordinateur, ou encore les assistants domotiques commandés à la voix ("*Google Home*" notamment).

II.2. Les réseaux de neurones artificiels

II.2.1. L'apprentissage machine

L'apprentissage machine, ou Machine Learning (ML) en anglais est un ensemble de techniques de l'IA qui rejoint la catégorie "*Penser comme un humain*" de la classification de Russel et Norvig et plus particulièrement "*Apprendre comme un humain*". En effet, le ML se caractérise par le fait que la machine ne connaît pas exactement les règles à suivre (l'algorithme) mais doit les apprendre à partir des données. Le ML est également la liaison entre l'algorithmique informatique, le traitement de données (avec label/complètes pour apprentissage ou sans/incomplètes pour analyse) et des caractéristiques (la sémantique des données, les objectifs à atteindre ...) [7, 13,14, 62]. Comme le montre la figure 4, le ML est très varié et peut être divisé selon le mode d'apprentissage (supervisé ou non), l'objectif de l'apprentissage (classification, clustering), ou encore les fait d'utiliser une seule technique contre la combinaison de plusieurs (méthode d'ensemble) :

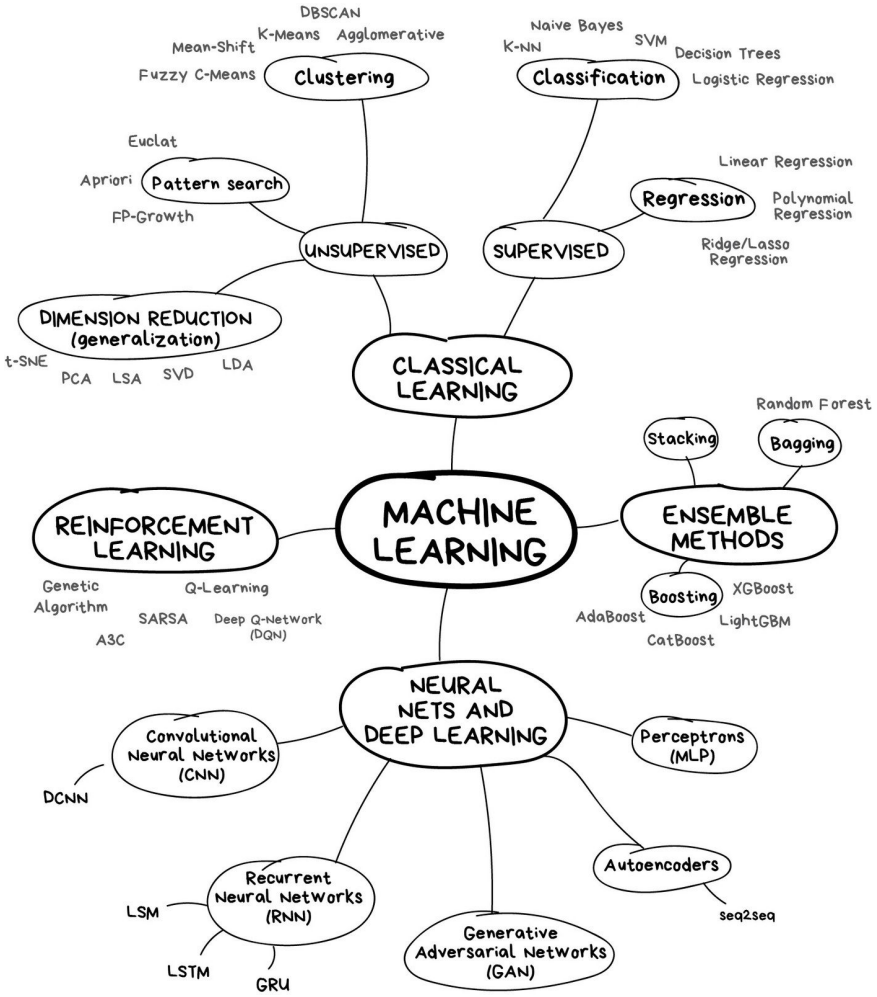

*Figure 4. Classification des différentes méthodes de l'apprentissage machine, extrait de [**13**]*

II.2.2. Les RNA

Les RNA sont un outil informatique basé sur un modèle apparu en 1950, à travers l'article fondateur "*What the frog's eye tells to the frog's brain ?*[7]", et faisant à la fois partie des statistiques et du ML (et donc de l'Intelligence Artificielle). Plus complexes que les méthodes du "*ML classique*", ils sont plus utilisés pour la résolution (classification ou régression généralement) de problèmes ardus, pour lesquels on n'a que très peu d'informations mais beaucoup de données brutes. En effet, les RNA se distinguent, comme nous allons le détailler par la suite, par une forte capacité d'apprentissage, une tolérance à l'erreur, un remarquable parallélisme et une forte non-linéarité. Techniquement, un RNA est un graphe pour lequel chaque noeud est un neurone formel et plus particulièrement, depuis 1958, un perceptron. C'est pour cela que l'on considère les RNA comme des "*perceptrons multicouches*" avec une couche d'entrée des données, une série variable de couches intermédiaires dites "*cachées*" et une couche de sortie du résultat [5, 7, 30, 31, 34].

La figure 5 présente justement le modèle du perceptron. On peut remarquer qu'il s'agit en premier lieu d'une série de valeurs d'entrées accentuées par un "*poids synaptique*" et regroupées ensembles par une fonction d'agrégation/recombinaison. Cette fonction représente le premier choix que l'on peut faire lors ce que l'on conçoit un ARN, mais généralement les deux plus utilisées restent

[7] *"Ce que les yeux de la grenouille disent à son cerveau"*

"*MLP*" (produit scalaire des vecteurs poids et entrées) et "*RBF*" (norme euclidienne des distances entre les entrées) [5, 30-32]. Ensuite, la valeur de cette fonction est comparée à un seuil d'activation en utilisant une fonction "*d'activation*". Cette fonction représente le second choix conceptuel à faire. Pareillement, il en existe plusieurs dont : "*ReLU*" et "*sigmoïd*" fortement recommandées. Plus récemment (2017), l'entreprise Google a proposé "*Swish*" qui semble combler les manques des deux autres et reprendre leurs avantages [5, 6, 9, 44-46].

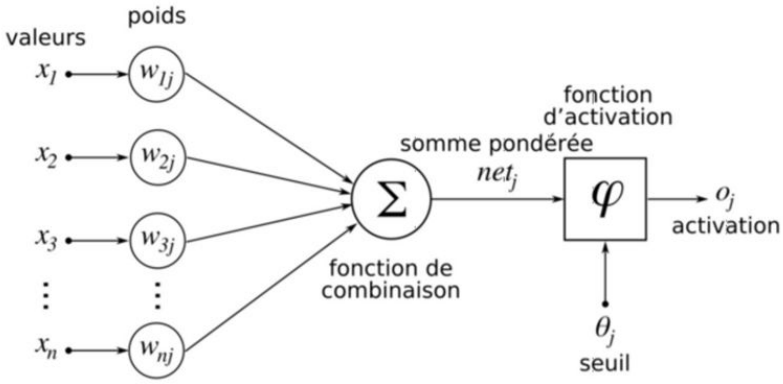

Figure 5. Le modèle d'un perceptron, extrait de **[31]**

Le choix conceptuel suivant est l'architecture globale du réseau. En effet, un seul neurone n'étant pas suffisant pour la non-linéarité mais également pour représenter de multiples sorties, on utilise généralement un graphe plus complexe. Ainsi le nombre d'entrées est lié au nombre de features que l'on dispose sur les données, le nombre de sorties doit être en accord avec les valeurs souhaitées et le nombre de niveaux cachés dépend généralement de la complexité du problème à résoudre (et la complexité des données). Il existe également des spécificités conceptuelles telles que d'accepter, ou non, la rétroaction d'un neurone sur lui-même. Finalement, il faut choisir les poids synaptiques initiaux et généralement ils sont attribués aléatoirement sachant qu'ils seront modifiés durant la phase d'apprentissage. Techniquement, il s'agit, comme pour tout graphe informatique, d'une simple matrice de valeurs [5, 30, 31]. La figure 6 présente un exemple de réseau complet pour un problème de classification binaire :

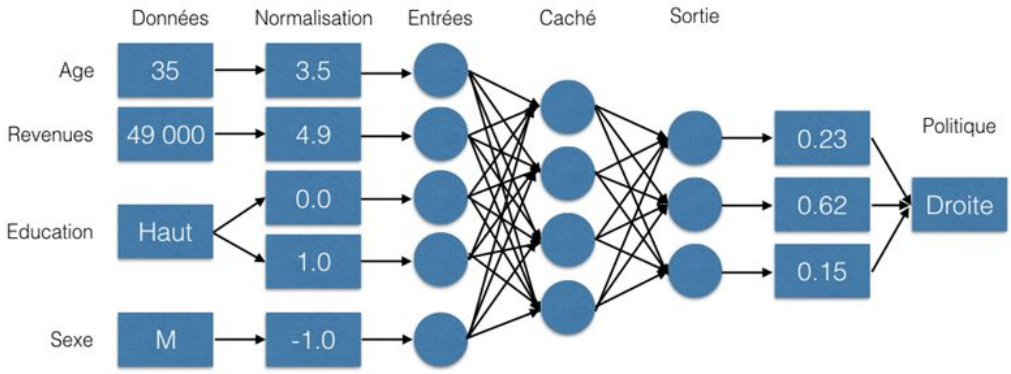

Figure 6. Exemple de réseau de neurones

L'utilisation d'un réseau de neurones se fait généralement à travers 3 grandes étapes : "*l'apprentissage*" à l'aide de données d'entraînement, les "*tests*" de l'apprentissage avec des données pour lesquelles on connaît également la sortie souhaitée ("*label*") puis finalement l'utilisation. Lors de l'utilisation on se contente d'un passage dit "*avant*" des entrées vers les sorties. Durant

l'entraînement, cependant, on fait des retours arrières pour modifier les poids synaptiques voire élaguer certains liens : c'est l'apprentissage ! Il s'agit alors d'un apprentissage dit par "*induction"* ou "*expérience"* [5, 35-38]. Pour cela, il existe de nombreuses méthodes basées sur des règles d'apprentissage telles que la "*règle de Hebb*". Cependant, les réseaux récents (à partir de 1974) utilisent généralement l'algorithme dit du "*gradient*" et la "*rétropropagation du gradien*t" et sont de par le fait appelés "*BPANN*[8]". Cet algorithme est une recherche locale pour laquelle on fait, à chaque itération, un recalcule des poids en commençant des sorties vers les entrées. Cette modification se base sur un calcul, à chaque neurone, du gradient (dérivée de la fonction d'activation) de l'erreur [5, 39-43]. Cet algorithme prend en considération plusieurs paramètres tels que le nombre d'itérations, la fonction dite de "*perte*" (fonction objectif) utilisée pour le calcul de l'erreur ou encore la fonction dite "*d'optimisation*" utilisée pour déterminer la valeur avec laquelle on va modifier un poids synaptique. Ces fonctions sont également connues, on peut citer "*Mean Squared Error*" pour la régression, "*Hinge*" pour la classification binaire ou encore "*Cross Entropy*" et "*KL divergence*" pour la classification multi-classes. Finalement il existe des techniques pour limiter le sur-apprentissage ou le blocage dans des "*minimums locaux*" telles que l'ajout de neurones dits "bias" ou la régularisation des fonctions de perte (on peut particulièrement citer "*L1*" et "*L2*" qui ajoutent les anciennes valeurs des poids dans l'équation) [5, 8, 47-49, 53-57].

Finalement, il existe une nouvelle branche de RNA dits "*réseaux profonds*" ("*Deep Learning*") qui ajoutent des couches pour le pré-traitement des données (réduction du bruit, détection des contours, réduction de la dimensionnalité) ou l'extraction des features. C'est le cas, par exemple, des réseaux "*convolutionnels*" utilisés pour le traitement de l'image et de la vidéo [50-52].

II.3. La méthode LDA

Latent Dirichlet Allocation ou "*LDA*" est un algorithme de réduction de dimensionnalité, et donc de ML classique, permettant de détecter les différents sujets (ou "*topics*") d'un corpus de documents et les différents mots clés qui les composent. Cet algorithme apparu en 2003 est à ce jour, l'algorithme le plus cité dans le domaine de l'Intelligence Artificielle. Bien que son fonctionnement soit relativement compliqué dû à l'aspect stochastique de l'algorithme et de son utilisation de métriques telles que TF/IDF, son utilisation reste assez simple. En effet, les seuls paramètres sont le nombre de sujets souhaités ainsi que le nombre de mots clés par sujets (ainsi que, parfois, des listes de mots vides lorsqu'il s'agit d'une application multilingue). Le résultat se présente, comme on peut le voir sur la Figure 7, sous la forme d'une liste de mots clés couplés à un pourcentage de représentativité face au sujet [10, 11, 12, 21] :

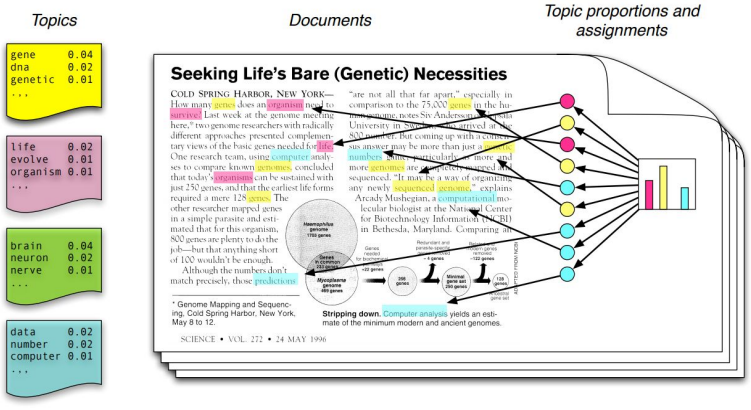

Figure 7. Exemple d'application de LDA, extrait de **[12]**

[8] *Back Propagation Artificial Neural Network*

II.4. La gamification

Bien que la définition puisse varier très légèrement d'un article à l'autre, la gamification, aussi appelée « *ludification* », correspond à l'application des mécanismes issus des jeux (et des réseaux sociaux) dans un contexte dit "*sérieux*" afin d'accroître la motivation de l'utilisateur [1, 2, 4]. Le jeu déclenche un plaisir à l'utilisation qui pourrait amener l'utilisateur à se fidéliser, à accroître sa fréquence d'utilisation ou encore à éprouver de l'intérêt pour le sujet. On retrouve dans la littérature particulièrement trois besoins qui sont comblés par les mécanismes du jeu : un besoin d'autonomie, de compétence et de relation [2, 4]. Dans [4], on découvre, en détail, les six mécanismes associés à six dynamiques (besoins) comme étant les principaux moyens à intégrer dans le processus sérieux pour le « *gamifier* » :

Mécanisme	Dynamique (besoin) liée
Système de points	La gratification
Différents niveaux	le statut
Des challenges	La réalisation
Systèmes de badges	La créativité
Un classement entre les joueurs	La compétition
Le don	L'altruisme

Figure 8. Les six mécanismes de la gamification

Cependant il existe des règles à respecter selon l'utilisateur ciblé. On retrouve dans différents travaux l'idée qu'une bonne conception de la gamification doit permettre d'obtenir ce que l'on appelle « *l'expérience ultime* » dans la théorie du « *Flow* ». C'est-à-dire ne rendre le processus ni trop ennuyeux ni effrayant (peur de perdre, de ne pas être à la hauteur, d'être rejeté...) [4] :

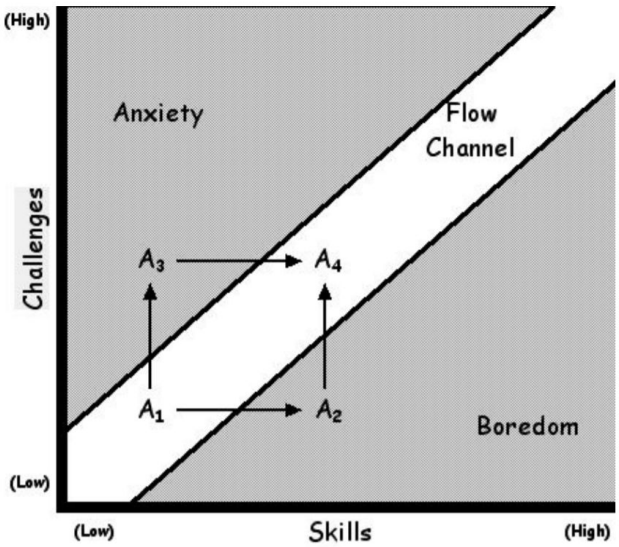

Figure 9. La théorie du Flow, extrait de **[4]**

III. Conception et implémentation du projet

La compréhension des notions présentées précédemment, des choix conceptuels et conseils à suivre ainsi que la mise en relation avec le problème posé, nous a permis de concevoir une solution adéquate. Comme nous allons le voir par la suite, cette conception suit les recommandations faites et en particuliers celles liées aux RNA et à la gamification. Cette troisième partie présente ainsi cette conception de la manière suivante : une conception générale du projet (fonctionnalités, utilisateurs, mécanismes généraux et modèle de données), suivie d'une conception du réseau de neurones et, finalement, une conception des différents algorithmes complémentaires permettant d'obtenir un projet complet.

III.1. Conception générale du projet

III.1.1. Architecture logique et physique du projet

Comme nous l'avons vu sur la Figure 2, nous avons choisi d'orienter ce projet sur l'application "*back-office*" du projet "*FlashCards*" et notamment le système de recommandation gamifié des cartes. L'étude des 3 concepts que sont LDA, l'apprentissage machine et les RNA accompagné de la gamification nous a permis de faire les choix conceptuels adaptés. Ainsi les RNA seront utilisés pour la classification des cartes en termes de niveau de difficulté, l'algorithme LDA permettra de détecter le thème sémantique d'une carte (information possiblement utile pour le réseau de neurones) et la gamification permettra de rendre le tout plus attractif à l'utilisateur.

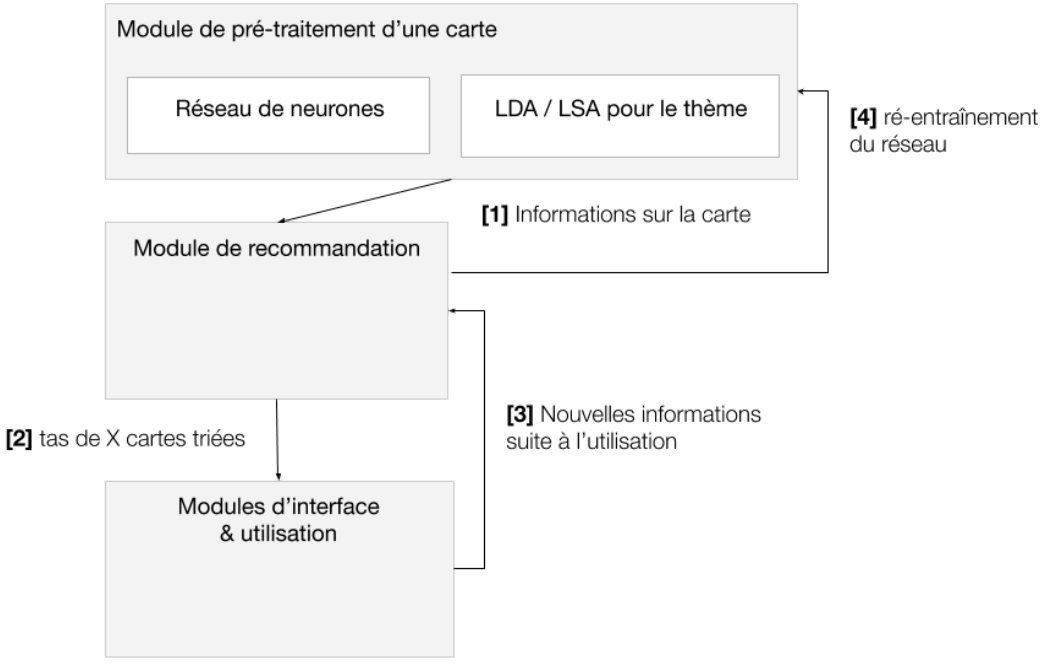

Figure 10. Architecture générale et modules du projet

Techniquement, nous avons décidé d'implémenter le projet en Python. En effet, la très grande majorité des librairies et algorithmes liés à l'IA sont dans ce langage. Ainsi, il existe une librairie, référence sur Github et Pip en Python pour LDA[9]. De la même manière, la librairie Google

[9] *Dépôts officiels de la librairie LDA sur PIP : https://pypi.org/project/lda/ et sur Github : https://github.com/lda-project/lda*

"*TensorFlow*" semble être la référence Python pour la création de RNA [61]. Python étant orienté objets depuis sa version 3.0, la modélisation et conception sera faite via UML. Finalement, nous avons également pu remarquer, sur la Figure 2, que le projet étant une application à part entière, il sera connecté avec l'application d'interface et la plateforme existante via des services web "REST". Pour réaliser cela, Python possède également une librairie qui a fait ses preuves : "*Flask*"[10].

Ensuite, nous avons commencé par concevoir "*l'architecture générale*" du projet, représentée sur la Figure 10, en termes de modules et interactions entre ces modules. En termes "*d'architecture physique*" cela se résume par 5 modules (et 5 fichiers pythons) : la couche d'accès aux données "*DAO*"[11], le système de recommandation en lui-même "*Engine*", l'algorithme "*LDA*", la couche de communication avec les applications externes "*Webservices*" et le réseau "*NeuralNetwork*". On peut alors remarquer une architecture dite "*n-tiers*" : traitements (Engine, LDA, NeuralNetwork), données (DAO) et affichage/communication (Webservices).

III.1.2. Interfaces, utilisateurs et cas d'utilisation

Bien que nous n'allons pas détailler l'application d'interface. Nous tenons simplement à dire qu'il serait souhaitable d'avoir un schéma de navigation le plus simple possible. Le point d'entrée serait une page de "*Wikimédica*" et donc directement le jeu ou une page de connexion et on pourrait ajouter une page "*leader board*" comme la gamification le recommande (voir Figure 8). L'application pourrait alors se contenter de trois pages uniquement ! Par ailleurs, ces pages devront être liées graphiquement au logo et les thèmes du site : donc Rouge/Rose/Gris et Noir.

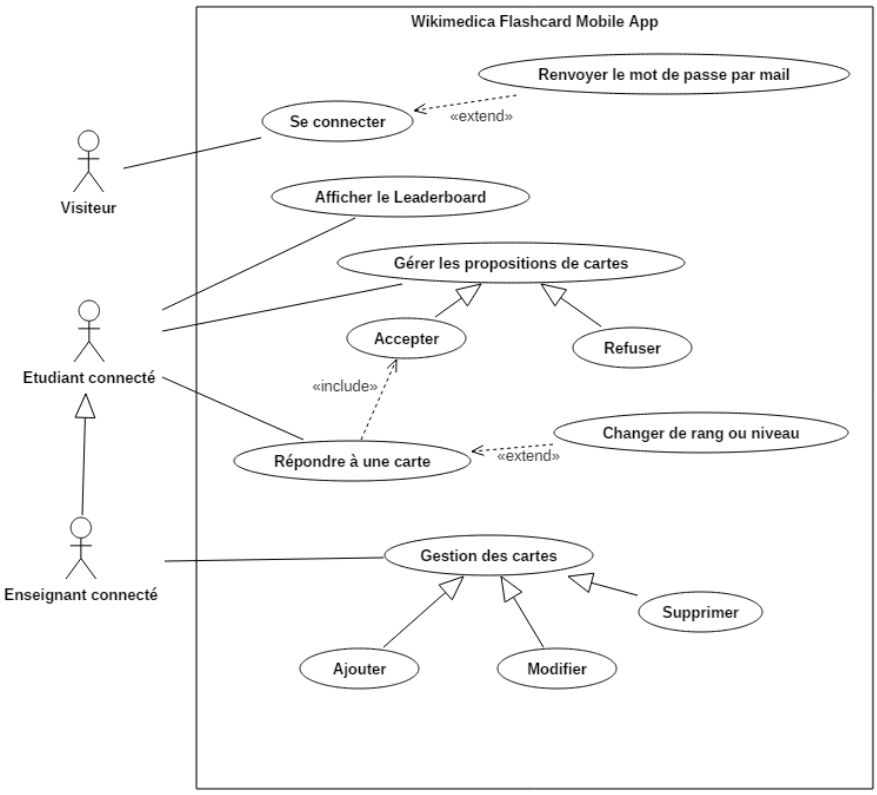

Figure 11. Utilisateurs et Cas d'utilisation du projet [UML]

[10] *Site officiel de la librairie Flask : https://www.palletsprojects.com/p/flask/*
[11] "*Data Access Objects*"

La figure 11 présente, quant à elle, les différentes fonctionnalités accessibles aux différents utilisateurs. Ce schéma est donc valable à la fois pour l'application d'interface mais également l'application "*back-office*" et les services qu'elle devrait fournir pour obtenir ces fonctionnalités. On peut remarquer alors 2 acteurs principaux une fois connectés : "*l'apprenant*" et le "*gestionnaire/enseignant*". La partie "intelligente" du système est surtout destinée à l'apprenant dans notre projet.

III.1.3. Conception et implémentation du modèle de données et de l'accès distant

Ensuite, nous avons pensé au module "*DAO*" et au modèle de données qui permettrait d'obtenir les fonctionnalités précédemment présentées. La figure 12 récapitule toutes les classes de données nécessaires à cela. On peut notamment remarquer la présence de niveaux (comme recommandé par la gamification) et thèmes ainsi que le fait de retenir en historique les différentes utilisations du système.

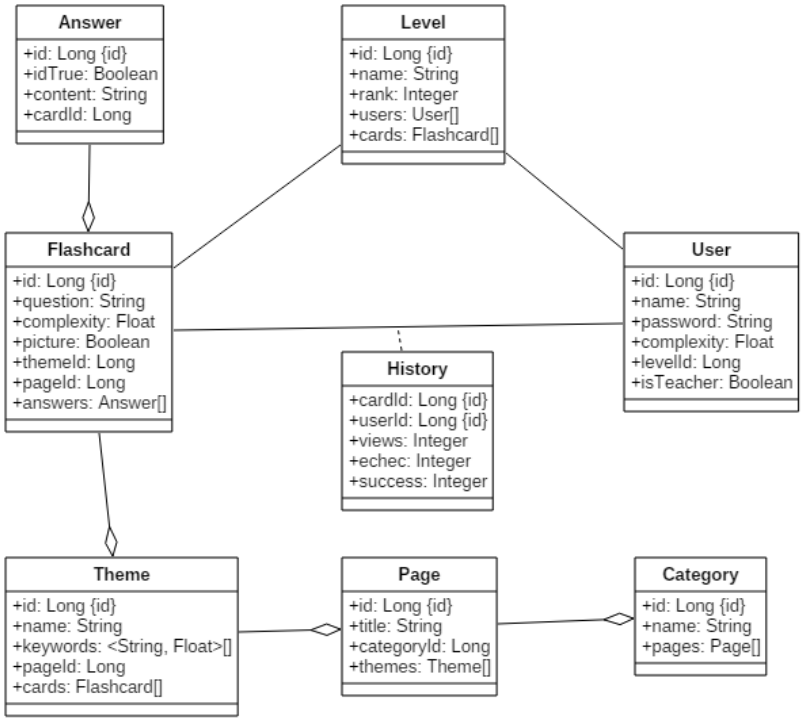

Figure 12. Diagramme de classe du projet [UML]

La figure 13 présente le code d'une classe de données, à savoir la classe FlashCard en Python. On peut y visualiser les différents attributs tels que prévus dans la conception de la Figure 12, ainsi qu'une méthode de "*sérialisation*" pour envoie à travers un service web.

```
# Une carte question
class FlashCard:
    def __init__(self, question, levelId=-1, picture="", answers=[], cardId=-1,
        themeId=-1, pageId=-1, complexity=0):
        self.id = cardId
        self.question = question
        self.levelId = levelId
        self.answers = answers
```

```
        self.complexity = complexity
        self.picture = picture
        self.themeId = themeId
        self.pageId = pageId

    def serialize(self):
        answersJson = list()
        for a in self.answers:
            answersJson.append(a.serialize())
        return {
            'id': self.id,
            'question': self.question,
            'picture' : self.picture,
            'levelId' : self.levelId,
            'complexity' : self.complexity,
            'themeId' : self.themeId,
            'pageId' : self.pageId,
            'answers' : answersJson
        }

    def __eq__(self, other):
        return self.id == other.id
```

Figure 13. Exemple d'implémentation de classe dans le module DAO

Afin de ne pas alourdir le projet par des configurations réelles, nous n'avons pas utilisé une réelle base de données mais plutôt de simples listes pré-remplies avec des données statiques. La figure 14 présente cette simulation d'une base à l'aide de listes :

```
# Definition de listes pour contenir l'ensemble des objets
flashcards = list()
users = list()
themes = list()
pages = list()
categories = list()
levels = list()
historic = list()

# Création des 10 niveaux de difficulté
levels.append(Level("Vraiment trop nul", 1, 1))
levels.append(Level("Débutant", 2, 2))
levels.append(Level("Facile", 3, 3))
levels.append(Level("Intermédiaire", 4, 4))
levels.append(Level("Avancé", 5, 5))
levels.append(Level("Difficile", 6, 6))
levels.append(Level("Hasardeux", 7, 7))
levels.append(Level("NP-complet", 8, 8))
levels.append(Level("Mission Impossible 007", 9, 9))
levels.append(Level("PHD", 10, 10))
```

Figure 14. Simulation de bases de données dans le module DAO

Encore une fois, les données et autres ressources sont vouées ici à être appelées à travers des services web REST et d'être utilisées dans une application tierce ou synchronisées avec la plateforme existante. Le tableau 1 récapitule l'ensemble des services et ressources qui ont été développés ainsi que des URI liées. La figure 15, quant à elle, présente un exemple d'implémentation (l'affichage du leaderboard d'utilisateurs) dans le module "*WebServices*".

```python
# Afficher l'ensemble des flashcards
@app.route('/flashcard', methods=['GET'])
def getAllCards():
    return jsonify([f.serialize() for f in flashcards])
```

Figure 15. Exemple de service d'accès distant (web) dans le module Webservices

Ressource et URI	Méthode HTTP	Description du service
Utilisateurs ['*/user'*]	GET	Afficher le leaderboard des utilisateurs
	POST	Connexion d'un utilisateur
Cartes ['*/flashcard'*]	GET	Afficher toutes les cartes avec leurs réponses
	POST	Ajouter une nouvelle carte
	DELETE	Supprimer une carte par ID
	PUT	Modifier les informations d'une carte
Réponses ['*/answer'*]	POST	Ajouter une réponse à une carte
	DELETE	Supprimer une réponse par ID
	PUT	Modifier les informations d'une réponse
Pages ['*/page'*]	GET	Afficher toutes les pages disponibles
	POST	Ajouter une nouvelle page
	DELETE	Supprimer une page par ID
	PUT	Modifier les informations d'une page
Catégories ['*/category'*]	GET	Afficher toutes les catégories disponibles
	POST	Ajouter une nouvelle catégorie
	DELETE	Supprimer une catégorie par ID
	PUT	Modifier les informations d'une catégorie
Jeu ['*/play'*]	GET	Récupérer un paquet de cartes pour jouer/essayer
	POST	Passer une carte (sans y répondre)
	POST	Répondre à une carte et vérifier les erreurs

Tableau 1.Récapitulatif des différents services web REST du module Webservices

III.2. Les réseau de neurones

Une fois que l'on a développé les tierces données et communication, il a fallu commencer les

traitements. Comme on l'a vu précédemment, nous en avons principalement trois types : le réseau de neurones, la recherche de thèmes avec LDA, et le système de recommandation "*Engine*" qui appelle et utilise les deux autres. La figure 16 présente ainsi notre réseau. Comme on peut le remarquer on a suivi les recommandations liées aux nombre d'entrées (elles correspondent aux informations et features que l'on peut obtenir), sorties (elles correspondent aux classes envisagées), niveaux intermédiaires (Limités à 2 car le problème ne traite pas de données complexes) et fonctions d'activation (favoriser *ReLu* et *Swish*). On remarque également que l'on s'attend à 10 niveaux de difficulté, ce qui correspond par ailleurs à la Figure 14.

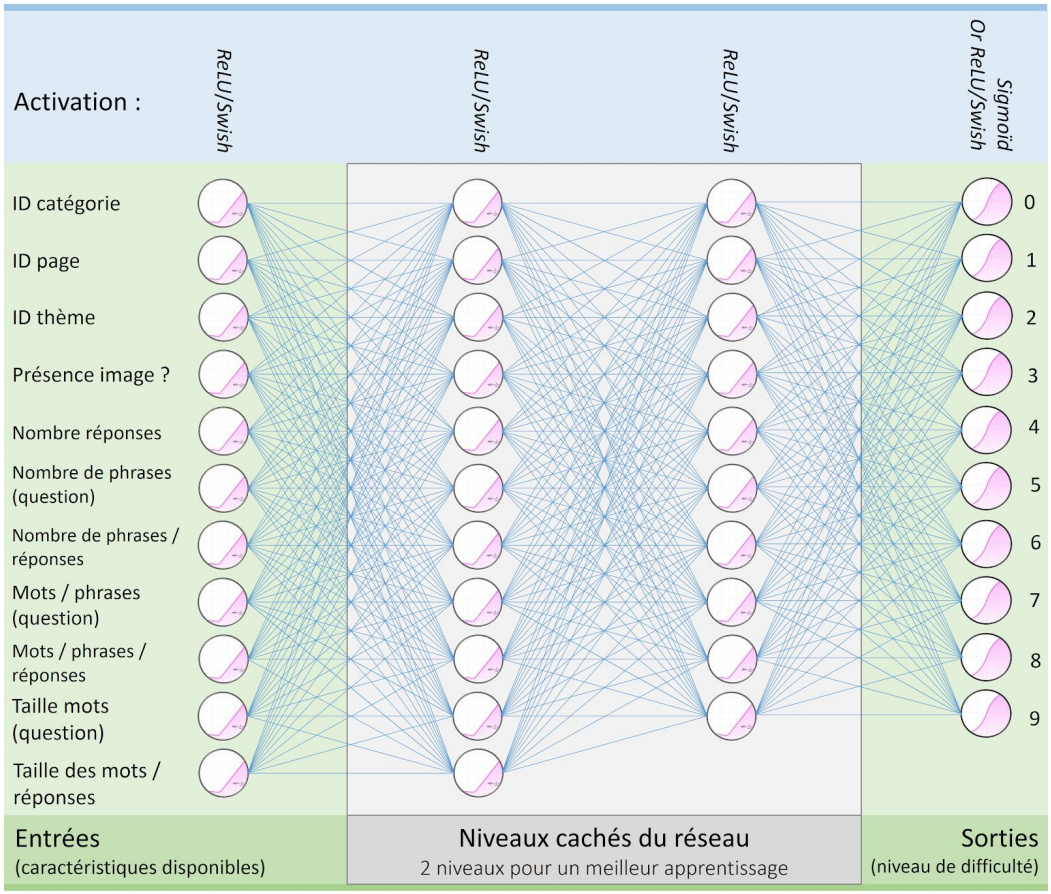

Figure 16. Conception complète de notre réseau de neurones

On peut également noter que les cinq premières entrées se retrouvent nativement sur le modèle de données (voir Figure 12) mais pas les cinq dernières. Ainsi, comme le montre, la Figure 18 nous avons dû ajouter des pré-traitements d'extraction de ces features :

Tout ceci sera valable à la fois pour les questions et les réponses (pour lesquelles une moyenne sera faite) :

nbr phrases = *Compter(séparation du texte par le caractère ".");*

nbr mots/phrases = *Moyenne les différentes phrases(Compter(séparation du texte par le caractère " "));*

taille des mots = *Moyenne les différentes phrases(Moyenne les différents mots(Compter le nombre de lettres));*

Figure 18. Obtention de plus de features pour le réseau de neurones

Voici également, sur la figure 17, le code du module "*NeuralNetwork*". On y découvre la fonction "*initNeuralNetwork*" qui crée le réseau tel que conçu sur la Figure 16, une méthode "*trainNeuralNetwork*" pour l'entrainer avec des couples "*entrée/label*" ainsi qu'une méthode "*classify*" pour utiliser finalement le réseau.

```
EPOCHS = 500

# Une fonction initiale pour l'initialisation de l'architecture du réseau
def initNeuralNetwork():
    model = tf.keras.Sequential([
      tf.keras.layers.Dense(11, activation=tf.nn.relu, input_shape=(11,)),
      tf.keras.layers.Dense(10, activation=tf.nn.relu),
      tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])
    model.compile(loss='categorical_crossentropy', optimizer='sgd',
    metrics=['accuracy'])
    return model

# Une fonction d'entrainement du réseau sur des données de test
def trainNeuralNetwork(model, inputs, labels):
    model.fit(inputs, labels, EPOCHS)

# Méthode principale du réseau de neurones (Classification)
def classify(inputs, model):
    return  model.predict(formatData(inputs))

# Formater les données pour tensorFlow
def formatData(inputs):
    data = list()
    data.append(inputs['categoryId'])
    data.append(inputs['pageId'])
    data.append(inputs['themeId'])
    data.append(inputs['picture'])
    data.append(inputs['nbrAnswers'])
    data.append(inputs['sizeAnswers'])
    data.append(inputs['sizeQuestion'])
    data.append(inputs['sizeSentencesQuestion'])
    data.append(inputs['sizeSentencesAnswers'])
    data.append(inputs['sizeWordsQuestion'])
    data.append(inputs['sizeWordsAnswers'])
    return data
```

Figure 17. Développement complet du réseau et fonctions liées

III.3. Algorithmes et traitements complémentaires au réseau

III.3.1. Traitements relatifs au calcul du niveau de complexité avec historique

Afin d'alléger les traitements, le réseau de neurones ne sera utilisé que lors de la création d'une nouvelle carte. A partir d'une certaine quantité d'historique, la mise à jour de la complexité et du niveau d'une carte se fera de la manière suivante :

On se base sur les données de tous les utilisateurs mais d'une seule carte !

$$\boldsymbol{rang} = \sum_{niveau-utilisateurs=1}^{10} \alpha * niveau\text{-}utilisateurs * \frac{Max\,(0;\, nombre\ d'échecs - 1)}{nbr\ total\ essais}$$

Figure 19. Calcul du niveau d'une carte selon son historique

Comme détaillé sur la Figure 19, on ne tient pas compte du premier essai mais on prend fortement en considération le niveau des utilisateurs dans le calcul (avec un facteur *alpha*). Par la suite, un niveau sera attribué à une carte et manière à répartir les rangs calculés sur 10 tranches de tailles égales.

III.3.2. Traitements relatifs à la composition d'un paquet de carte

Lors du déclanchement du jeu, le système de recommandation va également devoir choisir la composition en cartes et l'ordre des cartes dans le paquet généré. La Figure 20 présente cette composition et ordre selon l'historique, le niveau, la catégorie et page des cartes :

Composition des cartes proposées en termes de niveau (données d'un seul utilisateur) :

$$80\%\ niveau\_utilisateur + 10\%\ (niveau\_utilisateur + 1) + 10\%\ (niveau\_utilisateur - 1)$$

Composition des cartes proposées en termes de pages et catégories :

$$70\%\ de\ la\ page\ en\ cours + 20\%\ de\ la\ même\ catégorie + 10\%\ d'une\ autre\ catégorie$$

Formule de tri (données d'un seul utilisateur) :

$$\mathbf{rang} = \alpha * \frac{nombre\ de\ likes}{nbr\ total\ visualisations} + \beta * \frac{nombre\ d'échecs}{nbr\ total\ essais} - \gamma * nbr\_total\_visualisations$$

avec (données d'un seul utilisateur) :

$$\mathbf{nbr\_total\_essais} = nbr\_succès + nbr\_échecs$$

$$\mathbf{nbr\_dislikes} = nbr\_total\_essais - nbr\_total\_visualisations$$

Figure 20. Tri des cartes, pages, thèmes et catégories

```python
# Tri d'une pile de carte par intêret pour l'utilisateur
def orderCards(cards, userId):
    sortedCards = list()
    for c in cards:
        v = 0
        for h in getAllHistory(c.id, False):
            if h.userId == userId:
                v = ALPHA*(h.success + h.echec)/h.views + BETA*h.echec/h.views -
                GAMMA*h.views
                break
        sortedCards.append({'card' : c, 'value' : v})
    return sortedCards.sort(key=lambda card: card.value, reverse=True)

# Modifier le level d'une carte ou d'un user
def reloadLevelAndComplexity(elt, eltIsUser):
    listOfElements = users if eltIsUser is True else flashcards
    maxComplexity = 0
    result = {
        'complexity' : getComplexity(elt, eltIsUser),
        'level' : 10
    }
    for element in listOfElements:
        maxComplexity = max(maxComplexity, element.complexity)
    for i in range(10, 1, -1):
        if i/10 >= result['complexity']/maxComplexity:
            result['level'] = i
    return result
```

Figure 21. Exemple d'implémentation des formules en python dans le module Engine

La figure 21 présente un extrait de code du module "*Engine*" et notamment du tri des cartes dans un paquet.

III.3.2. Traitements relatifs à la recherche d'un thème après l'application de LDA

Finalement, tout comme l'utilisation du réseau a nécessité des prétraitements, l'utilisation de LDA et la recherche de thèmes nécessite des "*post-traitements*". En effet, comme on a pu le voir durant l'état de l'art (partie II.3), LDA permet de trouver les différents thèmes dans un corpus de cartes sous forme d'une liste de couples "*mot clé/pourcentage de représentativité*". LDA ne permet donc pas de décider quelle carte appartient à quel thème. Ainsi, comme détaillé sur la Figure 22, on a du ajouter un traitement qui se base sur la sortie de LDA. De plus, ce traitement permet de ne pas exécuter LDA (qui reste relativement "*lourd*") à chaque création d'une carte mais qu'à partir d'un certain nombre de modifications.

Dans l'équation suivante :
- η = nombre de thème (*pas utilisé ici*)
- φ = le nombre de mots par thème
- α_{ij} = le pourcentage choisi par l'application de LDA d'un mot *i* à un thème *j*.
- *k* = une flashcard

$$\boldsymbol{Theme_k} = Max_j\left(\sum_{i=1}^{\varphi} \alpha_{ij} * Présence(i,k)\right)$$

Figure 22. Choix d'un thème pour une nouvelle carte (après LDA)

Conclusion

A travers ce document, nous avons pu concevoir et implémenter, relativement rapidement, un premier projet qui utilise plusieurs outils de l'IA et l'apprentissage machine. Ce projet a également l'avantage de ne pas nécessiter de modification de la plateforme existante et de ne pas dépendre fortement de cette dernière. En outre, ce projet respecte les standards du "*génie logiciel*" plus classique tels qu'une architecture "*n-tiers*" et modulaire, une conception basée sur le standard UML ou encore un code documenté et maintenable.

Ce projet nous a permis par ailleurs de découvrir l'apprentissage machine (ML) et plus particulièrement deux de ses algorithmes mais également de revisiter le concept de gamification. En effet, comme on peut le voir à travers la "*Netographie*", la partie II est le résultat de très nombreuses lectures et recherches.

Cependant, ce projet a rencontré une limite évidente : le manque de données pour faire de l'entraînement ou du test des outils de l'apprentissage. Ces données sont pourtant essentielles à la réalisation de ces étapes. De plus, les concepts étant relativement nouveaux pour le concepteur, on peut remarquer une recherche de la simplicité et du "*réalisable*" plutôt que d'une idée révolutionnaire qui aurait pu se révéler infaisable dans le temps imparti.

Références

La gamification

[1] Alaa, A., Gary, B., Vanissa, W., & Ashok, R. (2014). Towards a Sustainable Gamification Impact. International Conference on Information Society.

[2] Aparicio, A., Vela, F., Sánchez, J., & Montes, J. (2012,October). Analysis and application of gamification. roceedings of the 13th InternationalConference on Interacción Persona-Ordenador. ACM, 17.

[3] Nassiri, A., Baudet, C., & Termins, F. (May 2017). De la compléxité de la notion de gamification à la compléxité de sa mise en oeuvre : une étude exploratoire dans une contexte d'application mobile touristique. 22eme colloque de l'AIM "Faire face à la compléxité dans un monde numérisé", Paris, France, AIM.

[4] Xu, Y. (2011). Literature review on web application gamification and analytics. Honolulu: HI,11-05.

L'intelligence artificielle et les réseaux de neurones

[5] Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application. Journal of microbiological methods, 43(1), 3-31.

[6] Karlik, B., & Olgac, A. V. (2011). Performance analysis of various activation functions in generalized MLP architectures of neural networks. International Journal of Artificial Intelligence and Expert Systems, 1(4), 111-122.

[7] Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering, 160, 3-24.

[8] Pereyra, G., Tucker, G., Chorowski, J., Kaiser, Ł., & Hinton, G. (2017). Regularizing neural networks by penalizing confident output distributions. arXiv preprint arXiv:1701.06548.

[9] Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. arXiv preprint arXiv:1710.05941.

L'algorithme LDA

[10] Andrzejewski, D., & Zhu, X. (2009, June). Latent dirichlet allocation with topic-in-set knowledge. In Proceedings of the NAACL HLT 2009 Workshop on Semi-Supervised Learning for Natural Language Processing (pp. 43-48). Association for Computational Linguistics. *https://pdfs.semanticscholar.org/8d70/3b10a63dea59bcdcf60499eb44ab657fffaf.pdf*

[11] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. Journal of machine Learning research, 3(Jan), 993-1022. *http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf*

[12] Bietti, A. (2012). *Latent Dirichlet Allocation*. mai 2012, working paper, http://alberto. bietti. me/files/rapport-lda. pdf. *http://alberto.bietti.me/files/rapport-lda.pdf*

Netographie

Le Machine Learning

Cours complets sur le Machine Learning :

[13] *https://vas3k.com/blog/machine_learning/*
[14] *https://medium.com/machine-learning-for-humans/supervised-learning-740383a2feab*

Les algorithmes du ML classique pour la classification :

[15] *https://en.wikipedia.org/wiki/Decision_tree_learning*
[16] *https://en.wikipedia.org/wiki/Naive_Bayes_classifier*
[17] *https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm*

Les algorithmes pour le clustering :

[18] *https://en.wikipedia.org/wiki/K-means_clustering*
[19] *https://en.wikipedia.org/wiki/Mean_shift*
[20] *https://en.wikipedia.org/wiki/DBSCAN*

Les algorithmes pour la réduction de dimensionnalité :

[21] *https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation*
[22] *https://en.wikipedia.org/wiki/Probabilistic_latent_semantic_analysis*
[23] *https://en.wikipedia.org/wiki/Principal_component_analysis*
[24] *https://en.wikipedia.org/wiki/Singular_value_decomposition*

Les algorithmes de règles d'association :

[25] *https://en.wikipedia.org/wiki/Association_rule_learning#Algorithms*

Les algorithmes d'apprentissage par renforcement :

[26] *https://en.wikipedia.org/wiki/State%E2%80%93action%E2%80%93reward%E2%80%93state%E2%80%93action*
[27] *https://en.wikipedia.org/wiki/Genetic_algorithm*

Les méthodes d'ensembles :

[28] *https://en.wikipedia.org/wiki/Random_forest*
[29] *https://en.wikipedia.org/wiki/Gradient_boosting*

Les réseaux de neurones artificiels

Réseaux de neurones artificiels :

[30] *https://en.wikipedia.org/wiki/Artificial_neural_network*
[31] *https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels*
[32] *https://fr.wikipedia.org/wiki/Perceptron_multicouche*
[33] *https://en.wikipedia.org/wiki/Feature_%28machine_learning%29*
[34] *https://missinglink.ai/guides/neural-network-concepts/complete-guide-artificial-neural-networks/*

Algorithmes d'apprentissage et apprentissage par inférence :

[35] *https://fr.wikipedia.org/wiki/R%C3%A8gle_de_Hebb*
[36] *https://fr.wikipedia.org/wiki/Inf%C3%A9rence_statistique*
[37] *https://fr.wikipedia.org/wiki/Inf%C3%A9rence_bay%C3%A9sienne*
[38] *https://fr.wikipedia.org/wiki/Apprentissage_par_renforcement*

Algorithme de rétropropagation du gradient :

[39] *https://fr.wikipedia.org/wiki/Gradient*
[40] *https://fr.wikipedia.org/wiki/Algorithme_du_gradient*
[41] *https://fr.wikipedia.org/wiki/R%C3%A9tropropagation_du_gradient*
[42] *https://missinglink.ai/guides/neural-network-concepts/backpropagation-neural-net*

works-process-examples-code-minus-math/
[43] *https://missinglink.ai/guides/neural-network-concepts/neural-network-bias-bias-neuron-overfitting-underfitting/*
Fonctions d'activation et choix d'une fonction d'activation :
[44] *https://fr.wikipedia.org/wiki/Sigmo%C3%AFde_%28math%C3%A9matiques%29*
[45] *https://engmrk.com/activation-function-for-dnn/*
[46] *https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/*
Régularisation L1 & L2 :
[47] *https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c*
[48] *https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261*
[49] *https://en.wikipedia.org/wiki/Regularization_%28mathematics%29*
Convolution, réseaux convolutifs et Deep Learning :
[50] *https://fr.wikipedia.org/wiki/Apprentissage_profond*
[51] *https://fr.wikipedia.org/wiki/Produit_de_convolution*
[52] *https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif*
Les fonctions de pertes (fonctions objectifs) dans un réseau de neurones :
[53] *https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/*
[54] *https://en.wikipedia.org/wiki/Mean_squared_error*
[55] *https://en.wikipedia.org/wiki/Hinge_loss*
[56] *https://en.wikipedia.org/wiki/Cross_entropy*
[57] *https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence*
Les Hypers-paramètres :
[58] *https://missinglink.ai/guides/neural-network-concepts/hyperparameters-optimization-methods-and-real-world-model-management/*
[59] *http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf*

Cours pratiques sur Python et TensorFlow

Cours complet sur le langage Python 3 :
[60] *https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-en-python*
Cours sur TensorFlow :
[61] *https://www.tensorflow.org/*
[62] *https://www.youtube.com/channel/UC0rqucBdTuFTjJiefW5t-IQ/videos*

Historique et définitions de l'Intelligence Artificielle

Généralités :
[63] *https://fr.wikipedia.org/wiki/Intelligence_artificielle*
[64] *https://en.wikipedia.org/wiki/Artificial_intelligence*
[65] *http://worrydream.com/refs/Minsky%20%20Steps%20Toward%20Artificial%20Intelligence.pdf*
Le modèle de Russel & Norvig :
[66] *http://www.chartsofhumanity.com/artificial-intelligence-1/*
[67] *http://www2.ift.ulaval.ca/~chaib/IFT-4102-7025/public_html/Fichiers/Acetates/Introduction.pdf*
Le workshop de Darthmouth :
[68] *https://en.wikipedia.org/wiki/Dartmouth_workshop*