

Mémoire de Master  
M2 I2A

Jean-Marc Gervais

Encadré par M. Jean-François Couchot, Maître de Conférences  
Institut FEMTO-ST, UMR 6174 CNRS – Département DISC – Équipe AND

13 juillet 2020

*Deep learning et respect  
des informations individuelles sensibles*

**Mise en œuvre pour la classification supervisée**

## Remerciements

Je tiens à remercier<sup>1</sup> M. J-F. Couchot, Maître de Conférences, qui m'a encadré dans le cadre de cette initiation à la recherche, pour ses conseils avisés et le temps conséquent qu'il m'a consacré, ainsi que pour la grande liberté qu'il m'a laissée dans les thématiques abordées. Je suis également reconnaissant envers Jean-François C. pour sa grande humanité et ses encouragements déterminants dans les moments difficiles. J'associe à ces remerciements M. Raphaël Couturier, Professeur des Universités, toujours réactif en cas de souci technique et qui m'a accueilli chaleureusement en tant que responsable de l'équipe AND. Les moyens techniques mis à ma disposition par cette équipe du département DISC et plus largement le laboratoire FEMTO-ST m'ont offert un cadre de travail très appréciable.

Je remercie aussi tout particulièrement mon épouse Florence, sans qui je n'aurais pas pu mener à terme ce travail, qui bien que modeste m'a largement occupé durant cette année. Sa patience, tous ses efforts pour m'offrir un maximum de temps disponible et les meilleures conditions de travail possibles ont été déterminants. Merci aussi à mes grands enfants Ludie et Arnaud, pour leur messages d'encouragement. Je n'oublie pas mes parents, qui m'ont donné il y a bien longtemps un cadre idéal pour m'épanouir dans mes études, dont je bénéficie encore aujourd'hui.

J'ajoute quelques mots pour mon « camarade de galère » Stéphane Robin, avec qui échanger régulièrement a été une source de réflexion intéressante et de motivation. Enfin, j'ai une pensée les enseignants chercheurs sans l'investissement desquels je n'aurais pas eu l'occasion de reprendre des études une fois plongé dans le monde du travail : ceux de l'université grenobloise (tout particulièrement Jean-Marc Vincent, Renaud Lachaize, et plus largement toute l'équipe qui s'est investie dans la liaison secondaire-supérieur, le DU-ISN et le DIU-EIL), puis ceux du Centre de Télé-enseignement Universitaire de Besançon, dont M. Fabien Peureux. Sa patience pour nous présenter la phase d'initiation à la recherche a été un élément important pour orienter au mieux nos choix.

---

1. Certains voient ces messages comme des figures de styles imposées. C'était mon cas il y a peu encore. Je les encourage à se lancer dans l'aventure, pour comprendre combien l'entourage est primordial et pourquoi on tient alors à en témoigner.

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Garanties de confidentialité</b>	<b>7</b>
2.1	Exemples de situations problématiques . . . . .	7
2.2	Premières modélisation de la confidentialité . . . . .	8
2.2.1	Échec des approches naïves . . . . .	8
2.2.2	$k$ -anonymat et notions connexes . . . . .	10
2.3	Confidentialité différentielle . . . . .	11
2.3.1	Aux origines : les réponses « randomisées » . . . . .	11
2.3.2	Les limites des perturbations aléatoires . . . . .	12
2.3.3	$\epsilon$ -indiscernabilité, $\epsilon$ -DP . . . . .	12
2.3.4	Sensibilité d'une fonction de requête . . . . .	17
2.3.5	Calibrage du bruit : mécanisme de Laplace . . . . .	17
2.3.6	Budget de confidentialité . . . . .	18
2.4	Variations autour de confidentialité différentielle . . . . .	21
2.4.1	$(\epsilon, \delta)$ -confidentialité différentielle . . . . .	21
2.4.2	Rényi-DP . . . . .	23
<b>3</b>	<b>Apprentissage automatique et confidentialité</b>	<b>26</b>
3.1	Deep Learning . . . . .	26
3.1.1	Neurone formel . . . . .	26
3.1.2	Organisation en couches . . . . .	26
3.1.3	Apprentissage supervisé . . . . .	27
3.1.4	Réseaux de neurones convolutifs . . . . .	31
3.1.5	Implémentation avec PyTorch . . . . .	33
3.2	Garantie de confidentialité différentielle . . . . .	34
3.2.1	Algorithme DP-SGD . . . . .	34
3.2.2	Exemple de MNIST . . . . .	35
3.2.3	Implémentation avec PyTorch-DP . . . . .	35
<b>4</b>	<b>Expérimentations et productions personnelles</b>	<b>41</b>
4.1	Calcul <i>a priori</i> du budget de confidentialité . . . . .	41
4.1.1	Écriture d'un script dédié . . . . .	41
4.1.2	Documentation (carnet Jupyter) . . . . .	42
4.1.3	Justifications des calculs sous-jacents . . . . .	46
4.2	Conversion à la DP de réseaux de classification . . . . .	48
4.2.1	Environnement de travail . . . . .	48
4.2.2	Expérimentations . . . . .	49
4.2.3	Résultats obtenus . . . . .	51
4.3	Analyse des besoins en mémoire sur le GPU . . . . .	54
4.3.1	Méthode générale . . . . .	54
4.3.2	Quelques précisions utiles . . . . .	55
4.3.3	Modélisation . . . . .	56
<b>5</b>	<b>Conclusion</b>	<b>59</b>
	<b>Bibliographie</b>	<b>60</b>
	<b>ANNEXES</b>	<b>62</b>

<b>6 Méthode RAPPOR – collecte des données</b>	<b>62</b>
6.1 Vue d'ensemble . . . . .	62
6.2 Algorithme fondamental . . . . .	62
6.3 Versions modifiées . . . . .	63
6.4 Influence des paramètres . . . . .	64
6.4.1 Epsilon . . . . .	64
6.4.2 Utilisation de cohortes . . . . .	64
6.4.3 Paramètres probabilistes et de configuration du filtre de Bloom	64
6.5 Confidentialité différentielle de RAPPOR . . . . .	64
6.5.1 Garanties sur la réponse randomisée permanente . . . . .	65
6.5.2 Garanties sur la réponse randomisée instantanée . . . . .	65
<b>7 Méthode RAPPOR – exploitation des résultats</b>	<b>65</b>
7.1 Étapes clés du décodage . . . . .	65
7.2 Évaluations expérimentales . . . . .	66
7.3 Limites de RAPPOR . . . . .	67

# 1 Introduction

Posons-nous la question suivante : « comment protéger les informations sensibles dans le contexte actuel de *big data* ? ».

Nul n'échappe au phénomène, que rien ne semble pouvoir arrêter à moyen terme. Enfin presque : même si nous laissons de côté ici la consommation de ressources et d'énergie, il semble qu'une bonne partie de la population commence à prendre conscience des dérives de la surveillance généralisée, qu'elle soit aux mains d'états (y compris dans les démocraties), ou de multinationales. La valeur de ses propres données et l'importance de les préserver commencent à être mieux comprises. Les citoyens pourraient donc résister à cette évolution, au niveau juridique bien sûr, mais aussi en se détournant des services qui ne fourniraient pas de garanties suffisantes. L'implication des GAFAM<sup>2</sup> au niveau de la recherche dans ce domaine semble d'ailleurs le confirmer. La bonne nouvelle est que leurs plateformes logicielles pour l'apprentissage automatique sont publiées sous des licences libres et que des initiatives voient le jour pour unir les efforts et construire des outils interopérables<sup>3</sup>. Ces problématiques sont éminemment actuelles, la recherche dans ce secteur est prolifique.

Bien entendu, il est impossible de traiter la question de manière exhaustive. Mais nous avons eu la chance de pouvoir l'aborder sous deux axes complémentaires :

- D'une part, nous avons questionné la notion de **confidentialité** : la « fuite » d'informations plutôt banales est-elle si problématique ? Et quand il s'agit de protéger certaines données, qu'entend-on par là ? Que doit-on, que peut-on préserver et comment le garantir durablement, formellement ?  
Nous verrons que si des solutions existent en matière de chiffrement, tout se complique lorsque l'on envisage de bénéficier du potentiel de collectes d'informations, sans pour autant nuire aux individus.
- D'autre part, nous avons eu l'opportunité de nous intéresser à **l'apprentissage automatique**, puis de passer à la pratique dans le cadre de la classification supervisée.  
Or les modèles basés sur les algorithmes de *deep learning* se nourrissent de quantités importantes d'informations. Il est avéré qu'ils peuvent ensuite laisser un utilisateur indélicat remonter aux données potentiellement sensibles qu'elles contiennent. C'est pour cela que les deux sujets se conjuguent parfaitement.

Dans ce document, nous partageons le résultat de notre travail dans l'ordre suivant :

- D'abord, nous faisons le point sur la réalité des risques en matière de divulgation d'information. Nous poursuivons par la présentation des principales formalisations envisageables de la notion de confidentialité et de leur intérêt, en se basant notamment sur des articles scientifiques récents. Nous montrons enfin qu'il est possible de quantifier la part de confidentialité qu'on doit accepter de perdre, quand on cède ou qu'on exploite des données.
- Dans la partie suivante, nous présentons quelques concepts-clés du *deep learning*, du neurone artificiel aux réseaux convolutifs. Nous montrons alors comment les mettre en œuvre à l'aide du récent *framework* PyTorch.  
Nous arrivons alors à la synthèse des deux domaines mentionnés, en exposant

---

2. Pour « Google Amazon Facebook Apple Microsoft »

3. Initiative OpenDP : <https://projects.iq.harvard.edu/opendp>

une méthode permettant gérer le compromis entre l'efficacité de tels outils et la protection des informations sensibles, concernant les données utilisées lors de l'apprentissage.

- Finalement, nous présenterons quelques réalisations et les expérimentations conduites dans ce cadre. Ce sera l'occasion de détailler et d'expliquer certains aspects. Mais également d'aborder quelques difficultés qui montrent la richesse des thèmes abordés.

Nous vous souhaitons autant de plaisir à lire ce document que nous avons eu à le rédiger.

## 2 Garanties de confidentialité

Si l'on entend encore trop souvent que l'on n'a « rien à cacher »<sup>4</sup>, il est important de prendre conscience que des informations apparemment anodines peuvent être discriminatoires, pour soi ou pour les autres. Pour garantir sérieusement leur confidentialité, il aura fallu définir avec rigueur cette notion. Ce sera l'objet des paragraphes suivants. Mais prenons d'abord quelques exemples concrets.

### 2.1 Exemples de situations problématiques

Retenons deux contextes qui attestent de la pertinence de notre problématique.

- Les terminaux mobiles comme les smartphones sont utilisés massivement comme supports d'**applications offrant un service personnalisé**. À cet effet, ces dernières doivent **analyser le comportement individuel** de l'utilisateur. Cela renforce la nécessité de **protéger les données personnelles**, spécialement celles **de localisation**, qui sont particulièrement sensibles, alors que peu d'utilisateurs en sont conscients.
- L'**apprentissage automatique** est utilisé dans un nombre grandissant de domaines. Il génère auprès du grand public pas mal de fantasmes. Pourtant, il n'est certainement ni une panacée, ni un démon omnipotent et invincible. Mais son développement exponentiel pose effectivement des questions auxquelles il est urgent de répondre. Le **risque en matière de divulgation d'informations** issues de celles qui ont servi à son entraînement n'est pas le plus évident, il est néanmoins **fondamental** car constitutif de la méthode.

Voici à présent quelques exemples d'extractions avérées d'informations sensibles. Si besoin, les références et détails sont donnés dans l'article de Vincent Primault *et al.* [PBBML19] dont nous tirons ces illustrations.

- Il est possible d'identifier des « **point d'intérêt** » à partir de traces de localisation, c'est-à-dire des lieux ou moments particuliers, porteurs d'une sémantique forte. Par exemple à partir de traces GPS de taxis, indiquant notamment les pauses, on a pu retrouver des adresses personnelles ou repérer les périodes de prières révélant la religion des chauffeurs avec une probabilité élevée. Avec l'évolution rapide des performances en apprentissage automatique, la découverte de tels points particuliers est encore facilitée.
- Des **relations sociales** peuvent être mises en évidence. En analysant les proximités temporelles et géographiques, avec des seuils bien choisis, des liens probables entre individus peuvent être établis. En y ajoutant la sémantique des lieux de rencontre, on parvient alors à caractériser ces relations. Des catégories peuvent être déterminées en utilisant un arbre de décision.
- On sait parfois déterminer l'identité d'un utilisateur à partir de ses traces de localisation (on parle de **ré-identification**). En effet, elles sont relativement uniques pour chaque individu : 4 points spatiaux-temporels suffisent généralement à caractériser une personne dans une foule. Le simple couple (lieu de travail ; habitation), ne serait-ce qu'indiqué à l'échelle du « bloc », est déjà fortement discriminant malgré l'importance en général des données temporelles, exclues ici. Dans le cas extrême où le graphe sous-jacent est vidé de ses données

4. Clin d'œil au documentaire « [NothingtoHide](#) ».

Voir également la page [Wikipedia](#) à propos de cette expression, ou <https://jenairienacacher.fr>

(celui des 10 lieux les plus fréquentés par exemple), sa topologie permet encore de retrouver une majorité d'individus !

Ajoutons que le matériel utilisé et les paramétrages des logiciels suffisent généralement déjà à signer un profil unique<sup>5</sup> et que les appareils connectés sont richement dotés en capteurs divers<sup>6</sup>, potentiellement bavards. La combinaison de ces informations rend donc la ré-identification assez facile, ce qui pose problème quand d'autres données sensibles sont divulguées simultanément. Nous y reviendrons à propos de la « pseudonymisation ».

- Enfin, analyser les habitudes ou les laisser modéliser par des algorithmes d'apprentissage automatique permet de **prédirer un prochain lieu de présence**. Cela a été illustré sur des points de *check-in* de Foursquare — dont la devise actuelle en page d'accueil est : « "où" est la clé »<sup>7</sup>. Certaines études montrent la possibilité de prévisions à plus longue échéance, en termes de lieu et de créneau horaire.
- Ces illustrations concernent surtout les applications embarquées sur des terminaux mobiles, mais l'**apprentissage profond** n'est pas épargné. Plusieurs études ont montré que les résultats de l'utilisation d'un réseau de *deep learning* permet parfois de remonter à des informations sur les données d'entraînement, potentiellement confidentielles. On parle d'**inversion de modèle**. Matthew Fredrikson *et al.* en donnent une illustration [FLJ<sup>+</sup>14] à propos d'un logiciel d'aide à la prescription personnalisée de Warfarine<sup>8</sup>. Il lui est possible de retrouver assez finement des marqueurs génétiques à partir de réponses fournies par cet outil et d'informations démographiques ou médicales librement accessibles par ailleurs. Florian Tramèr *et al.* [TZJ<sup>+</sup>16] montrent que même certains services de firmes de référence disposant de moyens conséquents au niveau développement, comme Amazon Learning Machine, peuvent rester vulnérables à ce type d'attaques.

Ce constat devrait suffire à se convaincre qu'il faut se préoccuper sérieusement de la protection de ce type de données. On peut être tenté de rejeter ces technologies. Mais de par leur richesse, les **données peuvent également être utilisées à bon escient**, pour analyser, optimiser ou inférer, notamment au bénéfice direct de l'individu qui les a fournies. Alors, que ce soit d'un point de vue éthique ou par intérêt commercial (pour ne pas perdre la précieuse confiance des clients), la confidentialité doit être respectée. En fait, nous verrons qu'il est nécessairement affaire de compromis entre le respect de la sphère privée et la diffusion de données suffisamment exploitables.

Pour quantifier la garantie offerte ou au contraire la perte de confidentialité, **des définitions précises seront nécessaires**. L'un des défis est de les rendre **lisibles pour l'utilisateur** non spécialiste, qui devrait être celui qui arbitre quand il s'agit d'utiliser ses données personnelles.

## 2.2 Premières modélisation de la confidentialité

### 2.2.1 Échec des approches naïves

Pour la première fois, en **1977**, le statisticien suédois **Tore Dalenius** définit précisément une garantie de confidentialité [Dal77] : en accédant à une base de don-

5. Voir <https://amiunique.org/fp> en ce qui concerne la navigation sur le web.

6. <https://epu2017.sciencesconf.org/data/Delabre.pdf> le détaille en pages 5-6.

7. <https://fr.foursquare.com/>

8. Un anti-thrombotique utilisé pour prévenir les attaques cardiaques

nées, l'adversaire ne doit pas être capable d'obtenir une information sur un individu qu'il n'aurait pas pu apprendre sans cet accès. Cynthia Dwork démontrera [Dwo06] que ce Graal reste inaccessible, notamment à cause de potentielles informations auxiliaires comme on l'a vu sur les exemples précédents. Cela va motiver la mise au point d'autres concepts moins rigides. Ce sera le début d'une belle aventure, avec une recherche prolifique, encore en pleine effervescence à l'heure actuelle. Nous partageons ici notre découverte des concepts qui ont émergé et de leur histoire.

La question de la protection des caractéristiques individuelles sensibles quand on stocke ou qu'on diffuse des données à des fins d'analyse globale s'est posée bien avant la vague de l'*open data* et le boom de l'apprentissage automatique (le premier recensement américain date de 1790 par exemple). Pour permettre des statistiques sur les habitants d'un état ou d'une région, il est tentant de fournir la base des données dans laquelle on a substitué des numéros aléatoires aux véritables noms (*par exemple en « hachant » ces derniers*) afin de gommer les identités réelles inutiles pour l'étude. Mais cette démarche, dite de **pseudonymisation**, s'avère comme son nom l'indique incapable de garantir l'anonymat et il reste souvent possible de retrouver un bon nombre, voire toutes les identités des individus enregistrés dans la base.

En effet, si un bon aléa ou un hachage robuste permettent en principe d'empêcher de retrouver les noms réels à partir des pseudonymes, c'est sans compter avec le recours à des bases d'informations auxiliaires : la faille tient aux données elles-mêmes, car on peut souvent en retrouver une partie dans d'autres bases et les ré-associer à un individu. On qualifie de **quasi-identifiant** ces *n*-uplets de champs présents dans des enregistrements d'une base et qui les caractérisent de manière unique, au moins avec une forte probabilité. Par exemple dans une table de patients hospitalisées, où la pathologie traitée constitue une information sensible, la donnée (âge approximatif ; poids approximatif ; origine ethnique ; statut marital) permet d'identifier des malades de manière quasi-unique [DN03]. Dans la partie précédente, nous avons mis en évidence que certaines données de localisation jouaient clairement ce rôle.

Plusieurs affaires célèbres l'ont également parfaitement illustré en situation réelle. Ainsi, en 2006, **AOL** publie<sup>9</sup> quelques 20 millions de requêtes effectuées sur son moteur de recherche par plus de 650 000 utilisateurs, après avoir *remplacé* les noms par des nombres aléatoires (afin d'étudier des liens éventuels entre les différentes requêtes d'une personne, ce qu'une simple *suppression* des identités ne permettrait pas). Suite à cela, certains citoyens ont pu être ré-identifiés (adresses, numéros de sécurité sociale, noms). En effet, nos recherches constituent souvent une « empreinte digitale<sup>10</sup> » unique.

Un autre cas marquant est celui de **Netflix**, qui diffuse en octobre de la même année les recommandations pseudonymisées de 500 000 clients. Au départ, il s'agit d'un prix d'un million de dollars à décerner au meilleur algorithme de recommandation. Mais Arvind Narayanan et Vitaly Shmatikov de l'université d'Austin au Texas ont montré [NS06] qu'ils pouvaient ré-identifier plusieurs profils, en exploitant notamment des données IMDB (*Internet Movie Database*). Cela vaudra une *class action* à Netflix<sup>11</sup> et un règlement amiable sans doute coûteux, ce qui rappelle au passage l'importance du respect de la sphère privée au delà de tout aspect éthique.

Ces différents exemples démontrent clairement que **la pseudonymisation est totalement insuffisante**. Ils ont également l'intérêt de rappeler qu'on ne peut pas se

9. Détails sur <https://www.nytimes.com/2006/08/09/technology/09aol.html>

10. Nous ne réitérerons pas ce jeu de mot, exploitant la traduction courante de « *digital* » par un adjectif qualifiant en principe ce qui a trait aux doigts plutôt que par « *numérique* », mais notons qu'ici les deux acceptations concordent de manière cocasse.

11. Voir [https://en.wikipedia.org/wiki/Netflix\\_Prize#Privacy\\_concerns](https://en.wikipedia.org/wiki/Netflix_Prize#Privacy_concerns)

fier à la seule intuition : **une protection rigoureusement démontrée est nécessaire**. Elle doit tenir compte de potentielles connaissances auxiliaires de l'adversaire, externes à la base sensible et par nature impossibles délimiter *a priori*.

### 2.2.2 *k*-anonymat et notions connexes

Voici une première parade à la ré-identification, qui se base sur l'agrégation de données, déjà en usage en statistiques. **Latanya Sweeney** de l'université d'Harvard l'a introduite en **2002** [Swe02] après une participation à une publication sur ce thème en 1998 [SS98], dans des articles où elle définit la notion de quasi-identifiant.

Sans rentrer dans un formalisme inutile ici, le *k*-anonymat (pour  $k \in \mathbb{N}^*$ ) est la **garantie que chaque *n*-uplet de quasi-identifiants quelconques est associé à *k* enregistrements** de la base **au minimum**, donc qu'un individu est protégé au sein d'un « groupe d'anonymat » d'effectif *k* ou plus. La probabilité d'identification individuelle tombe ainsi à  $\frac{1}{k}$  dans le pire des cas.

Concrètement, on doit d'abord identifier les regroupements d'attributs non sensibles susceptibles d'identifier une ligne de la base. Ensuite, on réduit le niveau de détail des valeurs de certains champs, jusqu'à pouvoir garantir qu'à chaque quasi-identifiant sont associés au moins *k* enregistrements : on peut regrouper des données numériques en classes, agréger des catégories. C'est la phase de généralisation. Prenons un exemple avec  $k = 2$  :

Nom	Âge	CP	Traitemen		Âge	Arrond.	Traitemen
...	19	25 000 <sub>B</sub>	Asthme	⇒	18-30	Besançon	Asthme
...	43	25 200 <sub>M</sub>	Cancer		18-30	Besançon	VIH
...	73	25 170 <sub>B</sub>	Leucémie		35-45	Montbéliard	Cancer
...	27	25 480 <sub>B</sub>	VIH		35-45	Montbéliard	VIH
...	35	25 120 <sub>M</sub>	VIH		60+	Besançon	Leucémie
...	61	25 000 <sub>B</sub>	Leucémie		60+	Besançon	Leucémie

Cependant, des **failles** subsistent : à partir d'un quasi-identifiant, il reste souvent possible de savoir que certaines valeurs sont exclues, pour un champ sensible (asthme impossible pour « 35-45, Montbéliard »), ou que d'autres sont plus probables que dans la population globale. Et en cas de valeur identique sur des données sensibles pour tous les membres d'un groupe d'anonymat, on retrouve même cette information de manière certaine (*leucémie*, pour « *60+, Besançon* »).

Pour éviter cette situation, le concept de ***ℓ*-diversité** ajoute la contrainte que tout groupe d'anonymat contienne au minimum *ℓ* valeurs distinctes des données sensibles à protéger. Pour accentuer l'indiscernabilité, on utilise parfois en plus la ***t*-proximité** : la distribution de chaque valeur sensible dans tout groupe d'anonymat doit alors rester suffisamment proche de celle observée dans la population globale. Dans ce dernier cas, on risque cependant de perdre l'intérêt même de l'étude des données, en gommant trop les corrélations...

L'un de ses intérêt majeurs du *k*-anonymat est qu'il est **compréhensible par un non-spécialiste**, comme l'utilisateur d'une application sur smartphone. En effet, c'est lui qui devrait accepter ou non de céder ses données, suivant les garanties qu'on lui fournit. Il doit donc pouvoir les cerner correctement.

Mais ce concept n'est **pas toujours simple à mettre en œuvre**, à automatiser. Comme on l'a vu, des failles peuvent subsister ; Il est difficile d'apporter des garanties formelles. Ce n'est pas rédhibitoire, comme le montre notamment son application concrète à la prédiction qualitative et quantitative d'interventions de

pompiers du Doubs<sup>12</sup>, par Jean-François Couchot *et al.* [CGR19]. Toutefois, les auteurs constatent dans ce cas précis qu'une autre approche, que nous allons justement aborder, donne de bien meilleurs résultats en termes de prédictions, tout en offrant de solides garanties concernant la confidentialité.

Il s'agit du concept devenu la référence en la matière et qui occupe toujours une place prépondérante dans la recherche actuelle. C'est lui ou ses variantes que nous mettrons en œuvre dans la suite, dans les applications pratiques.

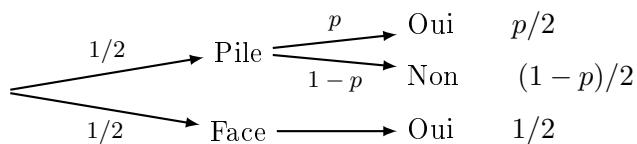
## 2.3 Confidentialité différentielle

En 2006, Cynthia Dwork chercheuse chez Microsoft, Frank McSherry, Kobbi Nissim et Adam D. Smith publient un article qui reste une référence, sur le concept de confidentialité différentielle [DMNS06]<sup>13</sup>. Cette publication leur vaudra le prix Gödel 2017<sup>14</sup>. La richesse de cette approche tient à son formalisme, aux démonstrations des propriétés garanties, indépendamment de toute connaissance *a priori* de l'adversaire.

### 2.3.1 Aux origines : les réponses « randomisées »

L'une des idées fondatrices est l'utilisation des réponses randomisées : ce mécanisme de protection introduit une part d'aléa entre la valeur réelle et celle qui est consignée ou diffusée. L'exemple de référence est l'utilisation de ce concept lors d'un sondage sur une question embarrassante (à propos de croyances, d'appartenance politique, de pratiques répréhensibles, etc.), pour laquelle on veut éviter que le sondé mente par crainte de se dévoiler. On le doit à Stanley L. Warner (1965) [War65].

En voici tout d'abord une version simplifiée. Considérons que c'est la réponse « Oui » qui peut être gênante. On demande à la personne interrogée de tirer secrètement à « Pile ou Face » : si elle tombe sur « Pile », elle doit répondre de manière honnête, sinon elle doit choisir le « Oui ». Cela lui garantit un **déni plausible**, concept clé dans ce cadre : rien ne permet d'être certain qu'une réponse « Oui » correspond à la réalité. Pourtant, on peut toujours estimer à partir du sondage la proportion  $p$  réelle de la caractéristique embarrassante en se basant sur le taux  $r \approx \mathbb{P}(\text{« Oui »})$  des réponses affirmatives (on montre aisément que  $p \approx 2r - 1$ ).

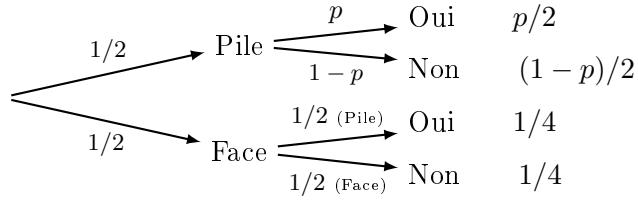


On peut objecter que le déni plausible n'est pas accordé à la réponse « Non », ce qui pourrait pousser à fournir cette dernière de manière malhonnête, pour s'assurer une solide couverture. On propose alors le raffinement suivant, le principe de base restant le même : le sondé tire deux fois à « Pile ou Face ». S'il obtient « Pile » la première fois, il répond de manière honnête, sinon, c'est le second tirage qui décide du résultat : « Oui » pour « Pile », ou alors « Non ». Les deux valeurs peuvent alors être mise en doute et l'on peut toujours estimer les véritables proportions dans le groupe sondé ( $p \approx 2r - \frac{1}{2}$ ).

12. Dans ce contexte, les données d'apprentissage devaient rester connues uniquement du service de secours, ne serait-ce que pour des raisons légales : dates, types et lieux d'interventions, identités des intervenants et des victimes.

13. Notons que ce terme n'apparaît pas dans cet article, où les auteurs parlent uniquement de l' $\varepsilon$ -indiscernabilité ( $\varepsilon$ -indistinguishability). Mais il s'agit du même concept.

14. <http://eatcs.org/index.php/component/content/article/1-news/2450-2017-godel-prize>



En fait, Warner fournit même une autre version, avec un tirage aléatoire non équitable, dans lequel dans un cas on pose la question « avez vous la caractéristique C ? » et dans l'autre la question contraire « est-il vrai que vous ne possédez pas la caractéristique C ? ». L'intérêt est de protéger encore mieux le secret du sondé.

En 2015 Graeme Blair, Kosuke Imai et Yang-Yang Zhou de l'université de Princeton ont recensé les applications pratiques de ce concept [BIZ15] et ont créé des outils permettant notamment des prédictions, comme avec les modèles de régression.

Mais nous allons voir que ces réponses randomisées à elles seules restent insuffisantes.

### 2.3.2 Les limites des perturbations aléatoires

Si la protection est solide pour une interrogation *unique* d'une personne donnée, elle s'amenuise clairement si on la réitère, dans le cadre d'un relevé quotidien par exemple. En effet, une analyse statistique permettra alors d'approcher la valeur réelle individuelle.

D'un point de vue formel, Irit Dinur et Kobbi Nissim ont montré en **2003** [DN03] que l'ajout d'un bruit important aux données d'une base était nécessaire pour garantir un minimum de confidentialité. En pratique, un tel bruit ruine la pertinence des requêtes. Plus précisément, en considérant une base assimilée à une chaîne de  $n$  bits, l'ajout d'une perturbation en  $\Omega(\sqrt{n})$  est nécessaire, sans quoi un adversaire peut en reconstruire en temps polynomial une très bonne approximation, à partir de requêtes renvoyant des sommes de bits (bruitées) sur un ensemble de positions bien choisies. La preuve repose sur un algorithme où ces sommes sont vues comme des codes correcteurs, qui utilise la programmation linéaire. Si l'adversaire n'est pas limité dans ses requêtes, le bruit doit même être linéaire en fonction de  $n$  pour assurer une garantie suffisante.

Dès lors, puisque la protection parfaite est illusoire si l'on veut tirer partie des données, le **besoin de quantification de ce qui est potentiellement divulgué** devient primordial.

### 2.3.3 $\varepsilon$ -indiscernabilité, $\varepsilon$ -DP

C'est en partant du constat que toute publication même mineure transgresse la définition de Dalenius [Dwo06], que Cynthia Dwork et ses collègues axent leur travail sur la quantification de l'information potentiellement divulguée. Ils proposent alors un mécanisme qui distord légèrement les valeurs sensibles avant de les communiquer. Ainsi, le résultat reste exploitable de manière statistique dans sa globalité, tout en préservant au mieux les informations individuelles. Il s'agit alors de gérer le compromis entre le niveau de perte de confidentialité et l'utilisabilité de ces données « bruitées ».

Notons que le terme lui-même de « confidentialité différentielle » s'inspire de la dérivation ou de la différenciation : on quantifie la variation maximale de la réponse à une requête, pour une petite modification de la base.

Considérons un mécanisme de réponse randomisée, qu'on note  $\mathcal{A}$ , défini sur  $D^n$  où  $D$  vaut  $\{0; 1\}^d$  ou  $\mathbb{R}^d$ . On peut modéliser l'image de  $\mathbf{x}$  par  $\mathcal{A}$  en tant que variable aléatoire. L'idée est de garantir que des bases très proches sont associées à des images par  $\mathcal{A}$  qui ont des distributions suffisamment semblables.

On utilise pour cela le nombre d'entrées qui diffèrent entre deux bases  $\mathbf{x}$  et  $\mathbf{x}'$  : c'est la distance de Hamming sur  $D^n$ , qu'on notera  $d_H(\mathbf{x}, \mathbf{x}')$ . Dire que deux bases  $\mathbf{x}$  et  $\mathbf{x}'$  diffèrent exactement d'une entrée revient donc à dire que  $d_H(\mathbf{x}, \mathbf{x}') = 1$ . On dit qu'elles sont **adjacentes**.

### Définition

Pour  $\varepsilon > 0$ , un tel **mécanisme**  $\mathcal{A}$  garantit l' $\varepsilon$ -**confidentialité différentielle**, qu'on abrégera en  $\varepsilon$ -**DP** (pour *Differential Privacy*)<sup>15</sup> si, pour toutes bases adjacentes  $\mathbf{x}$  et  $\mathbf{x}'$ , tout  $S \subseteq \mathcal{A}(D^n)$  et tout adversaire (*qui intervient dans le choix des requêtes*), on a<sup>16</sup>

$$\left| \ln \left( \frac{\mathbb{P}(\mathcal{A}(\mathbf{x}) \in S)}{\mathbb{P}(\mathcal{A}(\mathbf{x}') \in S)} \right) \right| \leq \varepsilon$$

Cela équivaut, compte tenu des rôles symétriques de  $\mathbf{x}$  et  $\mathbf{x}'$ , à

$$\mathbb{P}(\mathcal{A}(\mathbf{x}) \in S) \leq e^\varepsilon \mathbb{P}(\mathcal{A}(\mathbf{x}') \in S)$$

On dit également que  $\mathcal{A}$  est  $\varepsilon$ -**indiscernable** (formulation originelle de C. Dwork).

Le coefficient multiplicatif entre les probabilités aurait pu s'écrire plus simplement sous forme  $(1 + \varepsilon)$ , mais l'utilisation de l'exponentielle confère à cette définition de bonnes propriétés quand on enchaîne les mécanismes, comme on le verra. Et pour les petites valeurs de  $\varepsilon$ , on retrouve  $e^\varepsilon \approx 1 + \varepsilon$ .

Illustrons cette notion à l'aide de quelques cas particuliers.

- **Sondage randomisé** : reprenons l'exemple initial avec une réponse unique, « Oui » ou « Non ». Plaçons-nous dans le cadre de la seconde version, où un premier lancer aléatoire détermine si l'on doit répondre honnêtement, ou alors en se basant sur le résultat d'un second lancer. Une base  $\mathbf{x}$  est réduite ici à un unique enregistrement valant « Oui » ou « Non ». Un adversaire voudrait en connaître la valeur. Si c'est « Oui » (avec  $p = 1$  puisque la base est réduite à un unique élément), la probabilité que  $\mathcal{A}$  renvoie un « Oui » est  $\frac{3}{4}$ . Si c'est « Non », elle vaut  $\frac{1}{4}$ . Et bien sûr,  $\mathbb{P}(\text{Non}) = 1 - \mathbb{P}(\text{Oui})$ . Ici,  $d_H(\mathbf{x}, \mathbf{x}') = 1$  signifie qu'une base contient « Oui » et l'autre « Non ». Et les valeurs  $t$  produites par le mécanisme sont soit « Oui » soit « Non ». Si  $d_H(\mathbf{x}, \mathbf{x}') = 1$ , on a donc

$$\left| \ln \left( \frac{\mathbb{P}(\mathcal{A}(\mathbf{x}) = t)}{\mathbb{P}(\mathcal{A}(\mathbf{x}') = t)} \right) \right| = \left| \pm \ln \left( \frac{3/4}{1/4} \right) \right| = \ln 3$$

15. Nous utiliserons l'abréviation DP tantôt pour « confidentialité différentielle » (*Differential Privacy*), tantôt pour « garant de la confidentialité différentielle » (*Differentially Private*) en qualifiant un mécanisme randomisé. Il en sera de même avec les abréviations similaires qui suivront.

16. Dans sa formulation d'origine, C. Dwork note «  $\mathcal{A} = t$  » pour toute transcription  $t$  donc toute *issue*, plutôt que «  $\mathcal{A} \in S$  » pour tout événement  $S$ . Les deux formes sont équivalentes (Définition 1.1 [Vad17]). Mais la seconde est la seule adaptée pour les relaxations de la DP que nous allons aborder, c'est sans doute pourquoi elle s'est largement imposée. Notons enfin que l' $\varepsilon$ -indiscernabilité permet la comparaison de deux variables aléatoires, de manière plus générale.

Ce sondage randomisé est donc  $\varepsilon$ -indiscernable avec  $\varepsilon = \ln 3$ . Le quotient des probabilités qui interviennent dans la définition n'est autre que la cote utilisée pour les paris (3 : 1 ici). Par exemple dans le cas où la réponse fournie par le sondé est « Oui » :  $\frac{\mathbb{P}(\text{Vraie valeur} = \text{« Oui »} | \text{Réponse} = \text{« Oui »})}{\mathbb{P}(\text{Vraie valeur} = \text{« Non »} | \text{Réponse} = \text{« Oui »})}$

- **Sommes bruitées** : on considère une base  $\mathbf{x} \in \{0,1\}^n$ , donc réduite à  $n$  enregistrements valant 0 ou 1. On souhaite en extraire le nombre de 1, à savoir  $f(\mathbf{x}) = \sum_i x_i$ . On définit le mécanisme  $\mathcal{A}$  par  $\mathcal{A} = f + Y$ , où  $Y$  est une variable aléatoire qui suit la loi de Laplace  $Lap(1/\varepsilon)$ , de densité proportionnelle à  $h(y) = \exp(-\varepsilon|y|)$ .

Celui-ci est alors  $\varepsilon$ -indiscernable : en effet, comme  $|y'| - |y| \leq |y' - y|$ , on a  $\frac{h(y)}{h(y')} \leq \exp(\varepsilon|y' - y|)$ . De plus :  $\mathcal{A}(\mathbf{x}) = t \Leftrightarrow Y = t - f(\mathbf{x})$ .

Donc (avec un abus de notation entre distributions discrètes et continue)

$$\frac{\mathbb{P}(\mathcal{A}(\mathbf{x}) = t)}{\mathbb{P}(\mathcal{A}(\mathbf{x}') = t)} = \frac{h(t - f(\mathbf{x}))}{h(t - f(\mathbf{x}'))} \leq e^{\varepsilon|f(\mathbf{x}) - f(\mathbf{x}')|} \leq e^\varepsilon \quad \text{si } d_H(\mathbf{x}, \mathbf{x}') = 1$$

Ce mécanisme garantit donc la 1-DP.

- **Contre-exemple** : l' $\varepsilon$ -indiscernabilité peut s'avérer impossible pour certaines fonctions de requête, quel que soit le mécanisme associé.

En considérant  $f(\mathbf{x}) = \max_i(x_i)$  pour une base  $\mathbf{x} = (x_i)$  de  $n$  réels, en jouant par exemple sur la valeur de  $x_1$  dans  $\mathbf{x}$ , on peut obtenir deux bases Hamming-distantes de 1 telles que les images par  $f$  soient arbitrairement éloignées. Un mécanisme randomisé associé à  $f$  subira donc le même sort. Il peut donc produire une valeur donnée avec une probabilité nulle sur l'une des bases et non nulle sur l'autre. Cela rend le quotient des probabilités de la définition infini.

En général, on construit le mécanisme  $\mathcal{A}$  en partant de la fonction  $f$  de requête, qui associe un scalaire ou un vecteur de réels à la base utilisée, en lui additionnant un bruit aléatoire. Sa loi est choisie de manière à ce que  $\mathcal{A}$  donne les garanties d' $\varepsilon$ -DP nécessaires. L'originalité de cette notion est de considérer la situation **en fonction de la requête  $f$**  envisagée. Notons que  $f$  ne pourra pas viser d'action *individuelle* comme l'identification d'un terroriste dans la base, sans quoi la DP serait inapplicable<sup>17</sup>.

Le fait d'utiliser un facteur multiplicatif pour la DP **impose même aux petites probabilités de varier très peu** d'une base à une autre adjacente. C'est pourquoi c'est une définition **plus contraignante que la «  $\delta$ -proximité statistique »** utilisée en cryptographie ( $\max_S |\mathbb{P}(\mathcal{A}(\mathbf{x}) \in S) - \mathbb{P}(\mathcal{A}(\mathbf{x}') \in S)| < \delta$ , basée sur la distance en variations totales). En effet, on peut créer deux bases arbitrairement proches d'un point de vue statistique, mais violant la confidentialité différentielle : on fixe à zéro comme on vient de le voir la probabilité d'une valeur donnée dans une série et pas dans l'autre (réciproquement, une garantie d' $\varepsilon$ -DP implique la  $\delta$ -proximité statistique pour  $\delta = 1 - e^\varepsilon$ <sup>18</sup>).

Et nous verrons qu'il est souvent possible de fournir cette garantie de confidentialité différentielle en ajoutant un bruit relativement faible.

Cependant, la **sémantique** de l' $\varepsilon$ -DP est moins évidente qu'il n'y paraît. Il faut bien noter que la protection apportée est à l'échelle individuelle (on peut l'étendre dans une certaine mesure à un groupe d'individu, mais nous n'aborderons pas ce point).

---

17. Voir 1.6 dans [Vad17]

18. *Ibid.*

Prenons le cas suivant<sup>19</sup> : Gertrude, 65 ans, envisage de participer à une expérimentation pour la recherche médicale. Le concept l'intéresse, mais elle redoute un impact sur sa prime d'assurance sur la vie, à cause d'éventuelles informations personnelles qui seraient portées à la connaissance de son assureur (l'adversaire, en l'occurrence). Ce serait le cas par exemple, si les tests révélaient *chez elle* une probabilité de décéder dans l'année nettement supérieure à celle qui est connue pour les personnes ayant son profil.

Gertrude est assurée pour 100 000€ et la supposée probabilité qu'elle meure dans l'année est de  $\frac{1}{100}$ . Sa prime est donc fixée à 1 000€ (1% de 100 000€, on ne tient pas compte ici de la marge, ni des frais, etc.)

- Supposons que la publication de l'étude révèle que les adeptes du café comme elle, à cause d'une possibilité élevée d'attaque, ont deux fois plus de risque de décéder durant l'année, comparé à une femme quelconque au profil similaire. Si son assurance l'apprend, elle passera la prime de Gertrude à 2 000€. C'est beaucoup, mais la confidentialité différentielle ne la protège en rien ici. En effet, l'information n'est pas exclusive à Gertrude. Qu'elle s'abstienne ou non de participer à l'étude n'empêche pas l'obtention de résultats du même ordre, ni leur connaissance par l'assureur.
- Imaginons à présent que Gertrude décide de participer à l'expérimentation, qui révèle un problème cardiaque. En conséquence, on estime à 50 % sa probabilité de mourir durant l'année. Si l'assurance le découvrait, sa prime exploseraient à 50 000€ !

C'est là que la garantie d' $\epsilon$ -DP intervient : l'augmentation de la probabilité de décès pour Gertrude, estimée par l'assurance au vu des résultats publiés, ne pourra augmenter par rapport à ce qu'elle pouvait supposer au préalable que d'un facteur  $e^\epsilon$ . Pour une valeur initiale de 2 000€ et un paramètre de risque à  $\epsilon = 0,01$  la nouvelle prime ne dépassera donc pas 2 020€ environ ( $2\,000 e^{0,01} \approx 2\,000(1 + 0,01) = 2\,020$ ).

Gertrude peut être rassurée, pour sa cotisation en tout cas car la gestion de ses soucis de santé est d'un autre registre...

Abordons également le problème sous un autre angle. Le simple fait de pouvoir **dire si Gertrude a participé ou non** à l'expérimentation, ou qu'elle appartient ou non à une partie donnée des participants, pourrait avoir pour elle des conséquences délétères. En effet, nous avons déjà illustré la puissance de l'utilisation d'informations auxiliaires extérieures à la base, sur plusieurs exemples, pour inférer des informations sensibles.

La confidentialité différentielle permet de garantir qu'une réponse du mécanisme  $\epsilon$ -DP fournit un surplus d'information quant à cette appartenance qui reste faible.

Imaginons un mécanisme  $\epsilon$ -DP  $\mathcal{A}$  sur une base  $\mathbf{x}$ . Un adversaire voudrait savoir si la base utilisée par le mécanisme  $\mathcal{A}$  qu'il interroge contient une entrée pour Gertrude (on note  $\mathbf{x}_{in}$  une telle base), ou non (base  $\mathbf{x}_{out}$ ). L'état de ses connaissances avant de soumettre sa requête à  $\mathcal{A}$  peut être représentée par sa croyance *a priori*, autrement dit  $p = \mathbb{P}(\mathbf{x} = \mathbf{x}_{in})$ . Si l'adversaire n'a pas d'information particulière au départ,  $p = \frac{1}{2}$ . On voudrait déterminer un intervalle autour de  $p$  le plus réduit possible, contenant sa croyance après un appel à  $\mathcal{A}$ , au fait que Gertrude appartient à la base.

---

19. Les exemples sont empruntés à Kobbi Nissim, Thomas Steinke et Alexandra Wood *et al.* [NSW<sup>+</sup>17]

---

### Propriété<sup>20</sup>

Pour un mécanisme  $\varepsilon$ -DP noté  $\mathcal{A}$  sur  $x$ , si l'on note  $p = \mathbb{P}(x = x_{in})$  la croyance *a priori* d'un adversaire, quant au fait qu'un individu donné appartient à la base, et

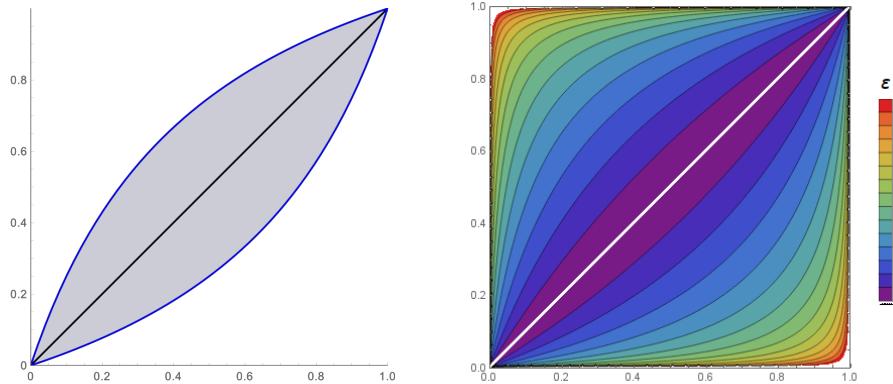
$p' = \mathbb{P}(x = x_{in} | \mathcal{A}(x) = y)$  sa croyance *a posteriori*,

à savoir après avoir obtenu une réponse  $y$  du mécanisme  $\mathcal{A}$  sur  $x$ , alors

$$\frac{p}{p + e^\varepsilon(1-p)} < p' < \frac{p}{p + e^{-\varepsilon}(1-p)}$$


---

Plus concrètement, cela signifie qu'on obtient, pour  $p'$  en fonction de  $p$ , à gauche pour une valeur donnée de  $\varepsilon$  et à droite pour une gamme de valeurs de ce coefficient :



Observons également quelques valeurs de la borne haute de la croyance *a posteriori* :

Croyance <i>a priori</i> (%)	croyance <i>a posteriori</i> maximale (%)							
	0,01	0,05	0,1	0,2	0,5	1	2	3
1	1,01	1,05	1,10	1,22	1,64	2,67	6,95	16,87
2	2,02	2,10	2,21	2,43	3,26	5,26	13,10	29,07
5	5,05	5,24	5,50	6,04	7,98	12,52	28,00	51,39
10	10,09	10,46	10,94	11,95	15,48	23,20	45,09	69,06
25	25,19	25,95	26,92	28,93	35,47	47,54	71,12	87,00
50	50,25	51,25	52,50	54,98	62,25	73,11	88,08	95,26
75	75,19	75,93	76,83	78,56	83,18	89,08	95,68	98,37
90	90,09	90,44	90,86	91,66	93,69	96,07	98,52	99,45
95	95,05	95,23	95,45	95,87	96,91	98,10	99,29	99,74
98	98,02	98,10	98,19	98,36	98,78	99,25	99,72	99,90
99	99,01	99,05	99,09	99,18	99,39	99,63	99,86	99,95

La propriété traduit le fait que l'évolution de la cote (au sens des paris, *betting odds*)  $\frac{\mathbb{P}(x=x_{in})}{\mathbb{P}(x=x_{out})}$  reste contrainte, du fait de l' $\varepsilon$ -DP :

$$\begin{aligned} \frac{\mathbb{P}(x = x_{in} | \mathcal{A}(x) = y)}{\mathbb{P}(x = x_{out} | \mathcal{A}(x) = y)} &= \frac{\mathbb{P}(x = x_{in}) \cdot \mathbb{P}(\mathcal{A}(x) = y | x = x_{in})}{\mathbb{P}(\mathcal{A}(x) = y)} \\ &\quad \times \frac{\mathbb{P}(\mathcal{A}(x) = y)}{\mathbb{P}(x = x_{out}) \cdot \mathbb{P}(\mathcal{A}(x) = y | x = x_{out})} \\ \frac{\mathbb{P}(x = x_{in} | \mathcal{A}(x) = y)}{\mathbb{P}(x = x_{out} | \mathcal{A}(x) = y)} &= \frac{\mathbb{P}(x = x_{in}) \cdot \mathbb{P}(\mathcal{A}(x_{in}) = y)}{\mathbb{P}(x = x_{out}) \cdot \mathbb{P}(\mathcal{A}(x_{out}) = y)} \\ \text{Soit } \frac{p'}{1-p'} &= \frac{p}{1-p} \times \frac{\mathbb{P}(\mathcal{A}(x_{in}) = y)}{\mathbb{P}(\mathcal{A}(x_{out}) = y)} \end{aligned}$$

---

20. La preuve et les illustrations qui suivent sont tirés de <https://desfontain.es/privacy/differential-privacy-in-more-detail.html>

$$\text{Or, d'après l'}\varepsilon\text{-DP} \quad e^{-\varepsilon} < \frac{\mathbb{P}(\mathcal{A}(\mathbf{x}_{in}) = y)}{\mathbb{P}(\mathcal{A}(\mathbf{x}_{out}) = y)} = r < e^{\varepsilon}$$

On conclut grâce à cet encadrement et à l'équivalence suivante (pour  $p, p' \neq 1$ ) :

$$\frac{p'}{1-p'} = \textcolor{violet}{r} \frac{p}{1-p} \iff p' = \frac{p}{p + \textcolor{violet}{r}^{-1}(1-p)}$$

### 2.3.4 Sensibilité d'une fonction de requête

Cynthia Dwork a cherché à caractériser les fonctions définies sur l'ensemble des bases de  $D^n$ , de manière à pouvoir déterminer une perturbation ajoutée suffisante pour obtenir un mécanisme  $\mathcal{A}$  qui soit  $\varepsilon$ -indiscernable.

#### Définition

La  $\ell_1$ -sensibilité d'une fonction  $f : D^n \rightarrow \mathbb{R}^d$  est le plus petit réel  $S(f)$  tel que, pour toutes bases  $\mathbf{x}, \mathbf{x}' \in D^n$  avec  $d_H(\mathbf{x}, \mathbf{x}') = 1$ , on a  $\|f(\mathbf{x}) - f(\mathbf{x}')\|_1 \leq S(f)$ .

Nous pouvons définir la sensibilité pour d'autres normes. Cependant, le «  $\ell_1$  » est souvent omis quand il n'y a pas d'ambiguïté.

Un point essentiel : cette notion est une **propriété intrinsèque** de  $f$ , indépendante de la base de donnée sur laquelle elle opère. C'est ce qui va permettre de **calibrer le bruit** ajouté et donc le mécanisme global uniquement **à partir de la requête**  $f$  envisagée. Dès lors, interroger une base complète se fera sans perte de confidentialité supplémentaire, par rapport à une demande identique sur un échantillon restreint. Reprenons quelques exemples :

- **Sommes bruitées** : on cherche à extraire d'une chaîne de bits  $\mathbf{x} \in \{0,1\}^n$  la valeur  $f(\mathbf{x}) = \sum_i x_i$ . La modification d'un enregistrement  $x_i$  d'une base à une autre peut modifier au maximum de 1 la valeur de  $f(\mathbf{x})$ , d'où  $S(f) = 1$ .
- **Histogrammes** : le domaine  $D$  est partitionné en  $d$  parties  $B_1, \dots, B_d$ . On considère la fonction  $f : D^n \rightarrow \mathbb{N}^d$  où chaque coordonnée de l'image, de  $k = 1$  à  $k = d$ , indique le nombre de valeurs dans  $B_k$ . Changer l'une des valeurs de  $D$  dans la base peut modifier au plus 2 coordonnées de son image (*une augmente de 1, une autre diminue d'autant, si la nouvelle valeur de départ n'appartient plus à la même catégorie*). La norme considérée est  $\ell_1$ , d'où :  $S(f) = 2$ .

Cette notion de sensibilité va prendre tout son sens en permettant de régler finement la perturbation aléatoire en fonction du type de requête, de manière à obtenir l' $\varepsilon$ -indiscernabilité.

### 2.3.5 Calibrage du bruit : mécanisme de Laplace

Voici un moyen de doser le bruit ajouter, suivant la requête :

#### Théorème

Pour tout réel  $\varepsilon > 0$  et toute fonction  $f : D \rightarrow \mathbb{R}^d$  de sensibilité  $S(f)$ , on définit  $\mathcal{A} : D \rightarrow \mathbb{R}^d$  par  $\mathcal{A} = f + Y$ , où  $Y$  est une variable aléatoire qui suit la loi de Laplace centrée <sup>21</sup>  $Lap_{\mathbb{R}^d}(S(f)/\varepsilon)$ .

Le mécanisme  $\mathcal{A}$  est alors  $\varepsilon$ -indiscernable.

<sup>21</sup>. Dans  $\mathbb{R}$ , la loi de Laplace (ou « double exponentielle ») centrée  $Lap(\lambda)$  a comme densité  $f(t) = \frac{1}{2\lambda} \exp(-\frac{|t|}{\lambda})$ . On l'étend ici à  $\mathbb{R}^d$  avec des composantes indépendantes et identiquement distribuées suivant  $Lap(\lambda)$ . La densité devient  $f(\vec{v}) = \frac{1}{2\lambda} \exp(-\|\vec{v}\|_1/\lambda)$

L'exemple précédent des sommes bruitées n'était qu'un cas particulier d'application de cette propriété.

Reprendons également le contre-exemple à propos de  $f(\mathbf{x}) = \max_i(x_i)$  pour une base  $\mathbf{x}$  de  $n$  réels. Ici, la sensibilité est infinie, donc le théorème ne s'applique pas. Ceci n'a rien d'étonnant puisque nous avions vu que l' $\varepsilon$ -DP est ici impossible.

Citons également quelques cas de fonctions à faible sensibilité, dont l'intérêt réside dans le peu de bruit à ajouter pour assurer la confidentialité différentielle :

- **Histogrammes** : on a montré que  $S(f) = 2$  pour la fonction  $f$  qui renvoie le vecteur des effectifs des enregistrements dans chacune des parties  $B_1$  à  $B_d$ , d'où un bruit indépendant de  $d$  suffisant, alors que  $S(f)$  augmentait en  $\Theta(\sqrt{d})$  en s'en tenant à un résultat établi précédemment [BDMN05]
- **Analyse disjointe** : on peut généraliser ce simple *décompte par classes*. Pour toute analyse renvoyant l'image par une fonction  $f$  de chaque partie  $B_i$ , qu'on note  $f(\mathbf{x})_i$ , on a  $S(f) \leq 2 \max_{1 \leq i \leq d} S(f(\mathbf{x})_i)$ , indépendamment de  $d$ .

Terminons sur une propriété qui montre bien la solidité du concept : la garantie concernant une information concédée par un mécanisme DP ne peut pas être dégradée par des opérations ultérieures sur cette donnée.

---

#### Propriété (robustesse au post-traitement)

Si un mécanisme randomisé  $\mathcal{A}$  sur une base  $\mathbf{x}$  est  $\varepsilon$ -DP, alors  
pour toute fonction  $f$  sur  $\mathcal{A}(\mathbf{x})$ , le mécanisme composé  $f(\mathcal{A})$  est  $\varepsilon$ -DP

---

### 2.3.6 Budget de confidentialité

Après avoir vu comment calibrer au plus juste un mécanisme isolé, intéressons-nous aux requêtes multiples. Dans un premier temps, prenons le cas où elles s'appliquent toutes à des parties distinctes de la base de donnée interrogée.

---

#### Propriété (« composition parallèle »)

Si les ensembles  $(D_i)_{i=1, \dots, k}$  sont des parties disjointes de la base  $\mathbf{x}$ , et si les mécanismes  $\mathcal{A}_i$  opèrent respectivement sur chacun des  $D_i$ , en garantissant la  $\varepsilon_i$ -DP, alors l'ensemble de ces mécanismes (présenté comme vecteur composé des retours de chacun, ou sous forme séquentielle) assure la  $(\max_i \varepsilon_i)$ -DP.

---

Remarquons que lors de l'apprentissage automatique que nous aborderons prochainement, le calcul de gradient bruité de la fonction de coût sur chaque *mini-batch*, autrement dit à chaque ensemble d'une partition du lot d'entraînement, correspondra à cette situation.

Par contre, comme on l'a noté précédemment (2.3.2), la protection due à l'ajout de bruit aux données brutes se dégrade en cas de répétition des requêtes sur l'ensemble de la base. Le résultat suivant permet de le quantifier la perte de garantie pour un tel enchaînement.

---

### Théorème (basique) de composition<sup>22</sup>.

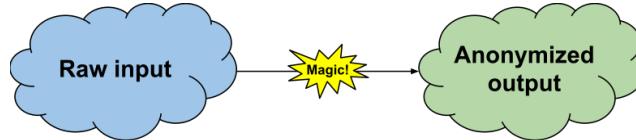
La composition séquentielle (*l'enchaînement*) de  $k$  mécanismes  $\mathcal{A}_i$  ( $i = 1, \dots, k$ ), qui assurent respectivement la  $\varepsilon_i$ -DP sur  $x$ , garantit la  $(\sum_{i=1}^k \varepsilon_i)$ -DP sur  $x$ .

---

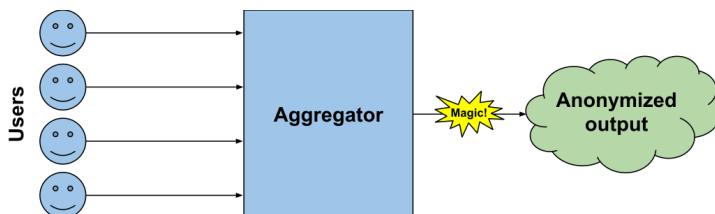
À noter que ce résultat reste valable pour une séquence adaptative, à savoir où l'adversaire peut choisir ses requêtes successives en fonctions des premières réponses données par le mécanisme.

Ainsi, pour une valeur de  $\varepsilon$  qu'un utilisateur s'est fixée au départ, chaque recours à un mécanisme entame *progressivement* le « **budget de confidentialité** » associé à  $\varepsilon$  : les possibilités d'interrogation s'amenuisent au fur et à mesure. Cette progressivité est une qualité essentielle de la DP. On paramètre donc le mécanisme en fonction de la requête  $f$  envisagée et de la perte de confidentialité tolérée.

Cette notion a deux conséquences majeures : d'une part, beaucoup de recherches vont concerner l'optimisation de la consommation de ce précieux budget  $\varepsilon$  qu'on s'est alloué au départ. D'autre part, on peut alors envisager **deux types de stratégies** pour communiquer des informations en respectant un certain niveau de DP, suivant le contexte. Illustrons de manière un peu caricaturale l'ajout de bruit comme une opération « magique » qui ajoute la protection attendue aux données traitées<sup>23</sup>.



- Dans une approche qu'on peut qualifier de globale, une entité sert d'**agrégateur de données brutes** (un service, une organisation, un gouvernement, etc.) C'est elle qui se charge de protéger ces données avant de les communiquer au monde extérieur.



Le point faible est clairement la confiance que l'utilisateur doit avoir en la partie centralisatrice. Sans compter qu'une faille, ou un acteur malveillant infiltré, pourraient exposer accidentellement l'intégralité des données sensibles. L'utilisateur n'a pas la possibilité de contrôler ce qui est réellement fait de ses informations. Les scandales répétés concernant les réseaux sociaux illustrent bien ces risques, même s'il n'y est pas question de sécurisation par ajout de bruit.

Par contre, l'utilisabilité des résultats tire profit de ce modèle, qui demande généralement une perturbation assez modérée (voir les fonctions à faible coût présentées précédemment).

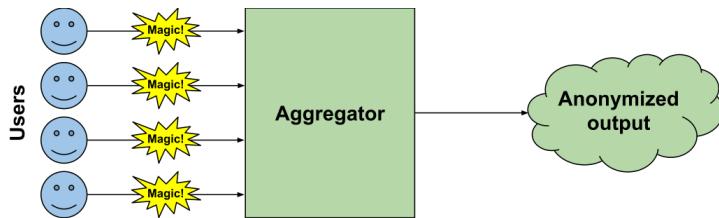
---

22. On le retrouve sous différentes formes dès les premières publications sur la DP, et I. Mironov le rappelle très clairement dans la partie II. de [Mir17]

23. Les illustrations sur ce thème sont tirées de <https://desfontain.es/privacy/local-global-differential-privacy.html>

On peut citer comme exemple concret de mise en œuvre l'*US Census*, chargé du recensement aux États-Unis d'Amérique.<sup>24</sup>

- Une alternative consiste à déporter au niveau local la DP. C'est l'utilisateur qui seul garde la main sur ses données et les bruite avant diffusion. L'éventuelle base agrégée peut alors publier des statistiques librement accessibles, puisqu'elles sont issues de données devenues non sensibles.



L'avantage de **ne plus reposer sur un tiers de confiance** est fondamental. On retrouve la situation initiale des réponses randomisées : seul le sondé connaît la réponse véridique à la question posée, ce qui n'empêche pas l'enquêteur de réaliser un sondage relativement fiable.

Inversement, comme le floutage intervient sur des données individuelles peu nombreuses, le bruit à ajouter est généralement plus important, ce qui dégrade davantage les informations transmises.

Cette approche est bien adaptée aux applications sur terminal mobile. Il faudra cependant veiller à ne pas consommer une part trop importante du peu d'énergie dont dispose en général ce type de dispositif. Et pour que la garantie soit opérante, l'utilisateur doit reporter sa confiance sur son appareil et son système d'exploitation. Dans ce domaine, il semble qu'aucun d'entre eux ne soit actuellement en mesure de donner de solides garanties, soit que son code reste fermé, soit que la structure de l'OS n'isole pas suffisamment les applications. Sans compter qu'ils reposent sur le *firmware* de l'appareil auquel on n'a pas accès<sup>25</sup>.

Un exemple intéressant est celui de RAPPOR<sup>26</sup>, l'outil implanté en 2014 par Google dans son Chrome. Il permettait de recueillir des habitudes de navigation, sujet sensible s'il en est. L'objectif final était d'améliorer le navigateur, en termes de fonctionnalités ou d'ergonomie, tout comme au niveau de la sécurité (détection d'utilisation malveillante). Nous avons étudié durant l'année cette méthode RAPPOR, dont nous fournissons une présentation en annexe 6. Apple a annoncé de son côté en 2016 qu'il utilise la confidentialité différentielle sur iOS 10. Le fait d'avoir été utilisé comme argument marketing<sup>27</sup> montre que l'utilisateur commence à être sérieusement attentif à cette problématique.

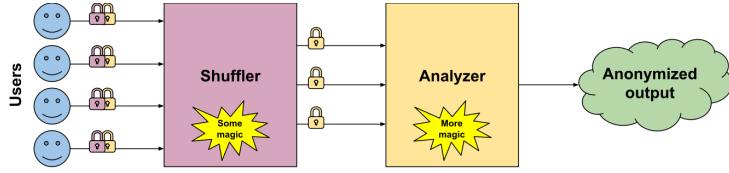
- Comme on peut s'y attendre, des approches intermédiaires tentent de bénéficier du meilleur des deux mondes. La démarche ESA (*Encode, Shuffle, Analyze*) est de celles-ci.

24. [https://www.census.gov/newsroom/blogs/random-samplings/2019/02/census\\_bureau\\_adopts.html](https://www.census.gov/newsroom/blogs/random-samplings/2019/02/census_bureau_adopts.html)

25. En tenant compte du fait que Framasoft n'est pas neutre et que R.M. Stallman n'est pas réputé pour ses propos nuancés, l'article <https://framablog.org/2011/10/10/android-stallman/> est intéressant et reste d'actualité. *La voie est libre, mais la route est longue* (devise de Framasoft).

26. *Randomized Aggregatable Privacy-Preserving Ordinal Response*, voir <https://github.com/google/rappor>

27. Savoir si l'application de la DP a été rigoureuse ou non est moins évident, voir <https://www.wired.com/story/apple-differential-privacy-shortcomings/>



L'utilisateur envoie des données protégées, qui sont ensuite pseudonymisées, mélangées et regroupées (on peut retrouver la  $k$ -anonymisation) par un agrégateur intermédiaire. Celui-ci les soumet à l'analyse statistique effectuée par une autre entité, dernière étape avant publication des résultats. La séparation en couches successives indépendantes augmente la garantie de confidentialité, de manière analogue à ce qu'on a pu constater au sujet de RAPPOR. Cependant, nous n'étudierons pas spécifiquement ce genre de configuration.

## 2.4 Variations autour de confidentialité différentielle

Diverses contraintes ont conduit à de nouvelles définitions pour qualifier la DP, généralement des relaxations de la version d'origine. Les principales caractéristiques sont cependant conservées : preuves formelles et indépendantes de l'attaque et des informations auxiliaires, robustesse au post-traitement, protection de l'information spécifiquement individuelle et dégradation progressive de la garantie pour une séquence d'appels.

Nous en présentons deux, qui seront celles mises en œuvre dans les applications étudiées dans la suite. La première est a été conçue pour être compatible avec mécanisme gaussien, autrement dit l'ajout à la requête d'un bruit qui suit une loi normale, celle que l'on rencontre « naturellement » en pratique. La seconde, plus élaborée, permettra de mieux préserver le budget de confidentialité lors des séquences d'appels aux mécanismes randomisés.

### 2.4.1 $(\varepsilon, \delta)$ -confidentialité différentielle

---

#### Définition $((\varepsilon, \delta)\text{-DP})$

Un mécanisme aléatoire  $\mathcal{A}$  défini sur un domaine  $\mathcal{D}$  garantit la  $(\varepsilon, \delta)$ -DP si pour toutes bases adjacentes  $x$  et  $x'$  de  $\mathcal{D}$  et tout événement  $S \in \mathcal{A}(\mathcal{D})$ ,

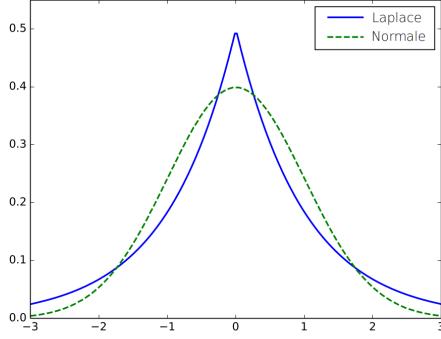
$$\mathbb{P}(\mathcal{A}(x \in S)) \leq e^\varepsilon \mathbb{P}(\mathcal{A}(x' \in S)) + \delta$$

On peut parler de confidentialité différentielle approchée, de paramètres  $(\varepsilon, \delta)$ .

---

L'ajout du terme  $\delta$  se traduit par une définition moins contraignante. Il lève la contrainte pour les probabilités les plus faibles, celles de la « queue de distribution ». La valeur de cette constante sera très petite en pratique, sans quoi la garantie perdrait tout son sens : si la base contient  $n$  éléments, elle sera choisie nettement inférieure à  $1/n$ .

La loi normale (ou de Gauss) s'observe dans beaucoup de situations, notamment quand il y a une répétition d'expériences aléatoires indépendantes identiquement distribuées. Une autre raison pour la préférer à celle de Laplace est sa convergence plus rapide vers 0 pour les valeurs éloignées de l'espérance (« queue de distribution serrée »).




---

### Définition

Le mécanisme gaussien  $\mathbf{G}_\sigma f$ , de paramètre  $\sigma$  appliqué à une fonction  $f$  définie sur  $\mathcal{D}$  et à valeurs dans  $\mathbb{R}^d$ , est défini par :  
 $\mathbf{G}_\sigma f(\mathbf{x}) = f(\mathbf{x}) + \mathcal{N}(0, \sigma^2 \mathbb{I})$  pour toute base  $\mathbf{x} \in \mathcal{D}$ ,  
où  $\mathbb{I}$  désigne la matrice identité dans  $\mathbb{R}^d$  :  $\mathbf{G}_\sigma f$  est l'ajout à  $f$  d'un bruit normal centré d'écart-type  $\sigma$  à chaque coordonnée.

---

Voyons en quoi cette définition s'accorde bien avec la  $(\varepsilon, \delta)$ -DP, alors qu'il est prouvé que la stricte  $\varepsilon$ -DP est impossible<sup>28</sup> avec  $\mathbf{G}_\sigma f$ , pour tout  $\varepsilon > 0$ .

---

### Définition

Comme annoncé, la  $\ell_2$ -sensibilité d'une fonction est définie comme pour la  $\ell_1$ , mais en utilisant ici la norme euclidienne  $\ell_2$  :  $\Delta_2(f) = \max_{\mathbf{x}, \mathbf{x}'} \|f(\mathbf{x}) - f(\mathbf{x}')\|_2$ .

---

### Propriété

Pour une fonction  $f$  définie sur un domaine  $\mathcal{D}$ , de  $\ell_2$ -sensibilité  $\Delta_2(f)$ , le mécanisme gaussien  $\mathbf{G}_\sigma f$  garantit la  $(\varepsilon, \delta)$ -DP quand  $\varepsilon < 1$  et

$$\sigma > \frac{\Delta_2(f)}{\varepsilon} \sqrt{2 \ln \frac{1,25}{\delta}}$$

(autrement dit quand  $\varepsilon < 1$  et  $\delta > \frac{5}{4} \exp(-\frac{[\Delta_2(f) \sigma]^2}{2})$ )

---

Remarquons que cette propriété ne fournit plus une valeur numérique unique pour quantifier la confidentialité, mais une infinité de couples  $(\varepsilon, \delta)$  qui conviennent.

On retrouve la compatibilité avec la composition : on dispose d'un « théorème de composition simple »<sup>29</sup> analogue à celui de la  $\varepsilon$ -DP : une séquence de  $k$  mécanismes respectivement  $(\varepsilon_i, \delta_i)$ -DP est  $(\sum_{i=1}^k \varepsilon_i, \sum_{i=1}^k \delta_i)$ -DP. Mais on a en plus un résultat qui permet en principe d'obtenir des résultats plus « serrés » :

---

### Théorème de composition avancé<sup>30</sup>

L'enchaînement séquentiel de  $k$  mécanismes aléatoires garants de la  $(\varepsilon, \delta)$ -DP respecte pour tout  $\delta' > 0$  la  $(\varepsilon', k\delta + \delta')$ -DP, où  $\varepsilon' = \varepsilon \sqrt{2k \ln(1/\delta')} + k\varepsilon(e^\varepsilon - 1)$ .

---

28. Partie II. dans [Mir17]

29. Théorème 3.16 dans [DR14].

30. Théorème 3.20 dans [DR14], repris dans le II. de [Mir17].

Là encore, on retrouve une infinité de couples pour quantifier la DP. On parle de « **courbe du budget** » en les assimilant à des coordonnées de points du plan. Les calculs deviennent vite complexes en pratique, si l'on part de  $k$  et du budget global à ne pas dépasser. Et ils sont vite inextricables, quand on cherche à composer des mécanismes  $(\varepsilon_i, \delta_i)$ -DP avec des coefficients différents, à cause de l'explosion combinatoire. Cela rend moins évident l'avantage théorique.

On n'a donc un outil qui n'est pas exempt de défauts, mais qui restera utile principalement pour deux raisons. Sa compatibilité avec le mécanisme gaussien, on l'a vu. Et sa **relative facilité d'interprétation**, en tout cas par rapport à la version suivante.

#### 2.4.2 Rényi-DP

**Martin Abadi et al.**, en **2016** [ACG+16], mettent au point la méthode du « comptable des moments » (*moments accountant*), qui apporte un gain important au niveau du budget de confidentialité lors d'une composition [ACG+16]. Ils parviennent à prouver cet avantage au niveau des bornes asymptotiques, mais aussi sur un exemple concret sur lequel nous reviendrons, car il est central en apprentissage automatique (il s'agit de la descente de gradient stochastique). Leur apport est fondamental.

**Ilya Mironov**, en **2017** [Mir17], reprend une approche équivalente au niveau théorique, mais reformulée à l'aide de la **divergence de Rényi**, qu'on notera **Rényi-DP** ou **RDP** dans la suite. C'est sous cette forme que nous présentons le concept, car elle est plus pratique à manipuler. Mais aussi parce que c'est elle que l'on retrouve au cœur du framework PyTorch-DP que nous utiliserons pour mettre en œuvre la confidentialité différentielle dans le cadre de l'apprentissage profond.

#### Définition

Soient  $P$  et  $Q$  deux distributions de probabilité définies sur un même espace  $\mathcal{R}$ .

Pour  $\alpha > 1$ , la **divergence de Rényi d'ordre  $\alpha$**  est définie par

$$D_\alpha(P||Q) = \frac{1}{\alpha-1} \ln \mathbb{E}_{x \sim Q} \left( \frac{P(x)}{Q(x)} \right)$$

(Cas fini  $\frac{1}{\alpha-1} \ln \sum_i \left( \frac{p_i^\alpha}{q_i^{\alpha-1}} \right)$  ou continu  $\frac{1}{\alpha-1} \ln \int p^\alpha q^{1-\alpha} d\mu = \frac{1}{\alpha-1} \ln \int \left( \frac{p}{q} \right)^\alpha dQ$ )

Et (prolongement par continuité du cas précédent)

$$D_1(P||Q) = \mathbb{E}_{x \sim P} \ln \frac{P(x)}{Q(x)}, \quad D_\infty(P||Q) = \sup_{x \in \text{supp } Q} \ln \frac{P(x)}{Q(x)}$$

La divergence de Rényi n'est pas symétrique, ne respecte pas l'inégalité triangulaire et n'est donc pas une distance. Nous pouvons tout de même garder l'idée qu'elle mesure un « écart » entre deux distributions.

#### Définition $((\alpha, \varepsilon_{\text{rdp}})\text{-RDP})$ <sup>31</sup>

Pour  $\alpha \geq 1$  et  $\varepsilon_{\text{rdp}} > 0$ , un mécanisme  $\mathcal{A}$  défini sur  $\mathcal{D}$  garantit

la **confidentialité différentielle de Rényi de paramètres  $(\alpha, \varepsilon_{\text{rdp}})$**  si pour toutes bases adjacentes  $x, x'$  de  $\mathcal{D}$ , on a :  $D_\alpha(f(x)||f(x')) \leq \varepsilon_{\text{rdp}}$ .

31. Nous avons pris le parti d'ajouter un « indice »  $_{\text{rdp}}$  au coefficient  $\varepsilon$ , pour le distinguer de celui utilisé dans les définitions précédentes et faciliter la compréhension, même s'il n'est pas présent dans les articles que nous avons pu consulter.

Cette définition reste bien plus difficile à interpréter que les précédentes. Tim van Erven et Peter Harremoës ont publié un article [vEH14] qui devrait aider à mieux cerner cette notion. Heureusement, nous allons voir que l'utilisation de ce concept est surtout utile au niveau comptable et qu'une compréhension incomplète de sa sémantique n'est pas rédhibitoire.

Retenons que la RDP est d'autant plus contraignante qu' $\alpha$  est grand. Dans le cas le plus exigeant, la  $(\infty, \varepsilon)$ -RDP est équivalente à l' $\varepsilon$ -DP. Elle implique toute  $(\alpha, \varepsilon)$ -RDP mais la réciproque est fausse pour  $\alpha$  fini.

On retrouve les propriétés essentielles communes aux mesures de confidentialité différentielle, parfois sous une forme moins stricte. La robustesse au post-traitement est confirmée bien sûr, sans quoi la notion perdrait son intérêt. Idem vis à vis des informations auxiliaires, même si ici c'est seulement l'espérance du rapport des cotés dont l'augmentation est bornée, et pas tous les cas possibles. La compatibilité avec les compositions séquentielles (y compris adaptatives là encore) s'exprime assez simplement, puisque pour un  $\alpha$  donné, on n'a qu'à « additionner les  $\varepsilon_{\text{rdp}}$  ». Plus formellement, on peut écrire :

---

#### Propriété (composition)<sup>32</sup>

Si  $f : \mathcal{D} \mapsto \mathcal{R}$  est  $(\alpha, \varepsilon'_{\text{rdp}})$ -RDP et si  $g : \mathcal{R} \times \mathcal{D} \mapsto \mathcal{R}'$  est  $(\alpha, \varepsilon'_{\text{rdp}})$ -RDP,  
alors le mécanisme définit par  $(X, Y)$  où  $X \sim f(\mathcal{D})$  et  $Y \sim g(X, \mathcal{D})$   
est également  $(\alpha, \varepsilon_{\text{rdp}} + \varepsilon'_{\text{rdp}})$ -RDP

---

Et comme on a là encore une infinité de couples  $(\alpha, \varepsilon_{\text{rdp}})$  pour un mécanisme donné, de multiples choix sont possibles sur une « courbe du budget ». Certains calculs resteront cependant assez simples. Ainsi, dans le cas du mécanisme gaussien, cette courbe est une simple droite.

La propriété suivante qui le montre n'est généralement citée que dans le cas de la sensibilité égale à 1. Nous avons choisi de l'énoncer dans un cadre plus général, afin de pouvoir justifier correctement les opérations effectués pour assurer la confidentialité différentielle dans le cadre de l'apprentissage automatique. C'est également parce que la sensibilité sera garantie par un « seuil de *clipping* »  $C$  dans ces algorithmes que nous avons opté pour cette notation.

Notons que nous avions quelques doutes sur la preuve de cette généralisation. Nous avons eu le plaisir de constater la disponibilité de M. Ilya Mironov, qui a confirmé notre raisonnement.

---

#### Propriété (RDP du mécanisme gaussien)<sup>33</sup>

Si une fonction  $f$  a une  $\ell_2$ -sensibilité de  $C > 0$ , alors le mécanisme  $\mathbf{G}_{C\sigma}f$  est  
 $(\alpha, \varepsilon_{\text{rdp}}(\alpha))$ -RDP pour tout  $\alpha > 1$ , si l'on prend  $\varepsilon_{\text{rdp}}(\alpha) = \frac{1}{2(C\sigma)^2}\alpha$

---

Il existe des résultats analogues pour la mécanisme de Laplace et d'autres, mais les courbes de budget sont plus compliquées et nous n'aurons pas à les utiliser pour notre travail sur le *deep learning*, c'est pourquoi ils ne seront pas mentionnés.

La RDP permet des économies sur la consommation du budget de confidentialité. Ce gain qu'apporte la RDP se fait au prix d'un concept peu explicite pour l'utilisateur.

---

32. Proposition 1 dans [Mir17].

33. Proposition 7, corollaire 3 dans [Mir17] pour la sensibilité 1. Confirmé pour le cas général en privé comme signalé, puis sur <https://github.com/facebookresearch/pytorch-dp/issues/11>

Fort heureusement, la propriété suivante va permettre d'associer idéalement RDP et  $(\varepsilon, \delta)$ -RDP, en affectant en quelque sorte la première au service comptabilité et la seconde au département communication. En effet, nous allons être en mesure de traduire le résultat optimisé grâce à la RDP en une formulation plus compréhensible en termes d' $(\varepsilon, \delta)$ -RDP.

---

**Théorème<sup>34</sup> (traduction de la  $(\alpha, \varepsilon_{rdp})$ -Rényi DP en  $(\varepsilon, \delta)$ -DP)**

Pour tout  $0 < \delta < 1$ ,  
si un mécanisme aléatoire est  $(\alpha, \varepsilon_{rdp})$ -RDP,  
alors il est également  $(\varepsilon, \delta)$ -DP pour  $\varepsilon = \varepsilon_{rdp} + \frac{\ln 1/\delta}{\alpha-1}$ .

---

Terminons par un cas particulier qui nous intéresse particulièrement car c'est un élément central en apprentissage automatique : le mécanisme  $\mathbf{SGM}_\sigma$  (pour *Subsampled Gaussian Mechanism*). Il s'agit d'un échantillonnage aléatoire d'une partie des données de la base, suivi d'une application de  $\mathbf{G}_\sigma f$ . On le réitère jusqu'à avoir traité l'ensemble des données.

Le calcul de RDP n'est pas immédiat et nous ne le détaillons pas ici. Mais l'idée globale est assez simple. Elle montre comme obtenir des dégradations les plus faibles possibles de la confidentialité, grâce à la RDP.

- On se fixe au départ une valeur de  $\delta$ , déterminée par le contexte (taille de la base, niveau d'exigence de confidentialité selon le type d'informations à préserver).
- Pour un certain nombre de coefficients  $\alpha$  bien choisis, on détermine  $(\alpha, \varepsilon_{rdp})$  pour notre mécanisme. On obtient ainsi des points régulièrement répartis de la courbe du budget. Cette discrétisation donne de bons résultats en pratique, pour un coût raisonnable.
- À l'aide de la propriété précédente, on calcule pour chaque point le coefficient  $\varepsilon$  associé, en termes de  $(\varepsilon, \delta)$ -DP.
- Pour notre comptabilité, on se base sur la plus petite des valeurs obtenues.

Ce tour d'horizon touche à sa fin. Il devrait permettre d'aborder sereinement la mise en œuvre de la confidentialité dans différentes situations. Nous allons justement nous intéresser à présent à l'un des contextes les plus courants, que nous mettrons pratique par la suite en y intégrant la *Differential Privacy*.

---

34. Proposition 3 dans [Mir17].

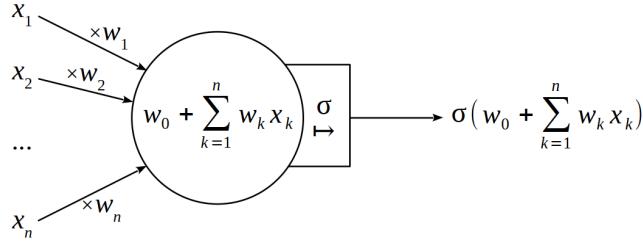
### 3 Apprentissage automatique et confidentialité

#### 3.1 Deep Learning

L'apprentissage automatique est un vaste domaine. Nos travaux durant l'année nous ont conduit à découvrir avec un grand plaisir des approches diverses, mais nous nous concentrerons ici sur des modèles de *deep learning* consacrés à la classification (par exemple d'images matricielles), en version supervisée : les catégories sont connues dès le départ, ainsi que les correspondances « image  $\mapsto$  catégorie » utilisées pour l'apprentissage. Nous nous limiterons à présenter quelques points essentiels, l'objectif étant d'être capable d'ajouter ultérieurement à ces réseaux neuronaux des garanties de confidentialité concernant les données d'entraînement.

##### 3.1.1 Neurone formel

Cette structure de base des modèles que nous allons construire est inspirée par la biologie. Elle a été introduite en 1943 par Warren McCulloch et Walter Pitts [LMMP59]. Les stimuli physiques sur les synapses sont remplacés par  $n$  valeurs numériques  $x_k$  constituant un vecteur  $\vec{x}$  en entrée. Ces données peuvent caractériser les différents pixels d'une image par exemple.



- Le neurone affecte à chaque nombre  $x_k$  un poids  $w_k$ , qui peut être adapté au fil du temps. D'abord, il en effectue la **somme pondérée** et ajoute un éventuel **biais** constant  $w_0$  — En considérant ce dernier comme une pondération associée à une entrée constante  $x_0 = 1$ , on peut ramener le calcul à un unique produit scalaire  $\vec{x}^\top \cdot \vec{w}$ , où  $\vec{x} = (x_k)_{k=0, \dots, n}$  et  $\vec{w} = (w_k)_{k=0, \dots, n}$ . Et si l'on préfère faire apparaître explicitement le biais  $b = w_0$ , on peut écrire  $\vec{x}^\top \cdot \vec{w} + b$  avec  $\vec{x} = (x_k)_{k=1, \dots, n}$  et  $\vec{w} = (w_k)_{k=1, \dots, n}$ .
- Ensuite, une **fonction non linéaire**  $\sigma$  dite **d'activation** (ou de sortie) est appliquée. Elle assure un effet de seuil (variation rapide quand la variable franchit une valeur donnée)<sup>35</sup>. Parmi celles qui sont couramment utilisées, citons la sigmoïde  $\sigma(x) = \frac{1}{1+e^{-x}}$ , la tangente hyperbolique  $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$  et la fonction ReLU (*Rectified Linear Unit*)  $\max(0; x)$  qui gomme les valeurs négatives.

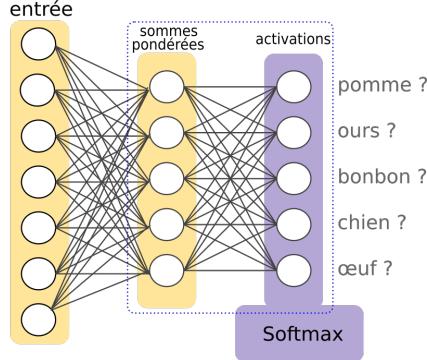
##### 3.1.2 Organisation en couches

Avec des paramètres bien choisis (pondérations synaptiques et biais), un neurone isolé est déjà capable d'assurer une classification à deux états, déterminés par le franchissement ou non d'un seuil par la valeur en sortie.

Pour trier en  $k > 1$  catégories, on doit former une couche de  $k$  neurones formels, prenant chacun le même vecteur  $\vec{x}$  en entrée. Le vecteur de sortie constitué des  $k$  images respectives de ces neurones s'écrit  $\vec{s} = \vec{x}^\top W$  (ou  $\vec{s} = \vec{x}^\top W + \vec{b}$  avec les biais à part), où la matrice  $W = [w_{ij}]$  de taille  $n \times k$  est formée des vecteurs  $\vec{w}$  de chacun

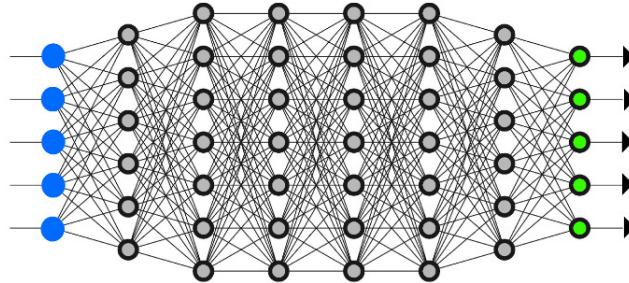
<sup>35</sup>. Pour plus de détails, voir cet article d'Ikram Chraibi Kaadoud sur <http://www.sciloggs.fr/intelligence-mecanique/repronons-bases-neurone-artificiel-neurone-biologique/>

des  $k$  neurones de la couche. Pour les problèmes de classifications on utilise souvent la fonction d'activation softmax qui transforme les scores (ou *logits*) obtenus en une distribution : sur chaque sortie  $m = 1, \dots, k$ , la probabilité estimée de la catégorie  $m$  vaut  $f_k(s_m) = \frac{\exp(s_m)}{\sum_{p=1}^k \exp(s_p)}$ <sup>36</sup>.



Enfin, pour améliorer les capacités du modèle, on peut « empiler » des couches successives de neurones, dont toutes les sorties sont respectivement connectées à toutes les entrées sur la couche suivante. On obtient ainsi un **perceptron multi-couches** (ou **MLP**, *Multi-Layer Perceptron*), capable de s'attaquer à des problèmes non linéaires, dont l'exemple typique du OU exclusif<sup>37</sup>.

Les données à analyser, par exemple les pixels d'une image, forment la couche d'entrée. Suivent des couches dites cachées et enfin celle de sortie donnant les scores des catégories. La profondeur dans « *deep learning* » fait référence au nombre de ces couches. On utilise fréquemment ReLU comme fonction de transfert interne.



Voilà pour la structure. Reste à aborder le sujet essentiel de l'ajustement progressif des poids synaptiques afin d'améliorer les performances de notre modèle.

### 3.1.3 Apprentissage supervisé

On utilise un jeu de données (*dataset*), constitué de nombreuses images (ou plus généralement de vecteurs) associés à des catégories ou « étiquettes ». La majeure partie servira à l'entraînement. Le reste est réservé aux tests. L'apprentissage supervisé consiste à confronter les valeurs produites par le modèle à partir des données d'entraînement aux sorties attendues, puis à modifier les paramètres du réseau neuronal en vue d'améliorer progressivement leur similitude, en réitérant cette étape de comparaison/ajustement. On teste la précision sur des données différentes de celles utilisées

36. Image adaptée partir d'une illustration du tutoriel suivant : <https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax>

37. On peut voir un problème de classification linéaire, dans un espace de dimension  $n$ , comme une situation où un hyperplan bien choisi peut servir de séparateur entre les entités à trier. Cette vision des choses est cependant dépendante de la représentation des données en entrée. Voir à ce sujet l'exposé et les animations de <http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/>

pour apprendre, afin de détecter un éventuel *overfitting* (sur-apprentissage<sup>38</sup>). Ce phénomène traduit la prise en compte par le réseau de détails trop spécifiques aux cas utilisés pour l'entraînement, ce qui induit une baisse des performances sur des entrées nouvelles.

Notons  $\vec{y} = (y_i)$  la réponse attendue. Ainsi,  $\vec{y}$  a toutes ses coordonnées à 0, sauf celle associée à la bonne catégorie qui est à 1. On utilise une **fonction de coût** (ou d'erreur, *loss function*) qui renvoie une « distance » entre  $\vec{y}$  le résultat calculé par le modèle, tous ses paramètres étant dans un état donné. L'apprentissage consiste à minimiser cet écart.

Détaillons la méthode utilisée<sup>39</sup>. Pour simplifier à ce stade, supposons qu'à chaque étape, on soumet au modèle une entrée unique.

En reprenant des conventions usuelles, nous noterons comme un exposant entre parenthèses le niveau de la couche, en partant de 0 pour les entrées, soit :  $x_i = a_i^{(0)}$ . Ensuite,  $a_i^{(L)}$  est la valeur en sortie du neurone d'indice  $i$  (de 1 à  $n_L$  pour des raisons pratiques) sur la couche  $L$ . En notant  $\sigma$  la fonction d'activation, on a donc

$$a_j^{(L)} = \sigma(z_j)^{(L)} \quad \text{avec} \quad z_j^{(L)} = \sum_{i=1}^{n_{L-1}} w_{ji}^{(L)} a_i^{(L-1)} + b_j^{(L)}$$

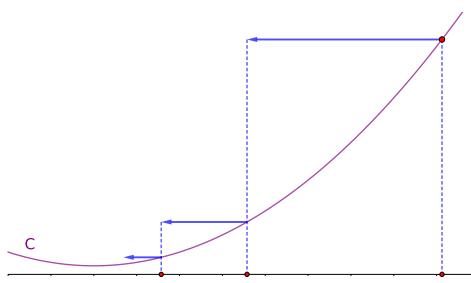
si  $b_j^{(L)}$  représente le biais et  $w_{ji}^{(L)}$  le poids de la synapse allant du neurone d'indice  $i$  de la couche  $(L-1)$  à celui d'indice  $j$  du niveau  $(L)$ . *Attention à l'ordre des coefficients, choisis en cohérence avec ceux de la matrice associée au calcul de cette partie linéaire.*

Enfin, supposons qu'on utilise comme coût la somme des écarts quadratiques :

$$C = \sum_{k=1}^{n_{L_{\max}}} (a_k^{(L_{\max})} - y_k)^2$$

On cherche à appliquer la méthode de **descente de gradient** (*deepest descent*) aux paramètres du modèle, afin de minimiser l'erreur  $C$  en sortie. Il s'agit de soustraire à chacun, de manière itérative, la dérivée partielle en ce point du coût par rapport à ce paramètre, multipliée par un « taux d'apprentissage »  $\lambda > 0$  donné. Par exemple pour  $w_{ji}^{(L)}$ , (idem pour  $b_j^{(L)}$ )

$$w_{ji}^{(L)} \leftarrow w_{ji}^{(L)} - \lambda \frac{\partial C}{\partial w_{ji}^{(L)}}$$



38. La traduction littérale par « surajustement » me semblerait plus judicieuse, mais ça ne semble pas le terme habituellement utilisé.

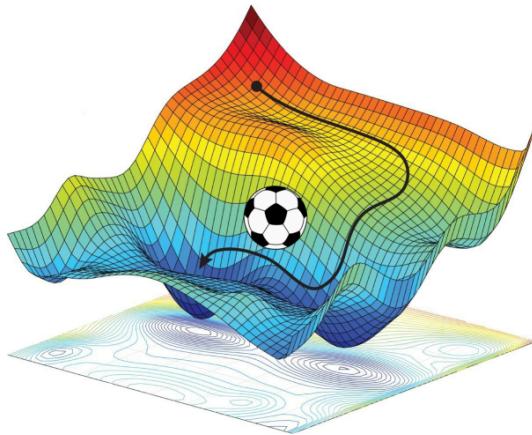
39. On trouve de multiples présentations du principe de *backward propagation*, mais les versions précises semblent plus rares. Or, une bonne compréhension sera importante lors de notre analyse de la consommation de mémoire en phase d'apprentissage, d'où ces détails. Les vidéos YouTube de 3blue1brown seront un excellent complément <https://frama.link/backprop>

Notons que la convergence vers le minimum global n'est assurée que pour les fonctions convexes. En extrapolant le schéma précédent, on peut également réaliser qu'un taux d'apprentissage trop faible ralentit la convergence alors qu'une valeur trop importante risque d'engendrer des « oscillations » autour du minimum, vers lequel on n'arrive plus à tendre.

Au final, en considérant les paramètres comme les coordonnées d'un vecteur  $\vec{\theta}$ , cette descente de gradient se traduit globalement par l'application à chaque étape, en notant  $\vec{\nabla}$  l'opérateur gradient dont les composantes sont les dérivées partielles, de l'affectation

$$\vec{\theta} \leftarrow \vec{\theta} - \lambda \cdot \vec{\nabla} C(\vec{\theta}) \quad (1)$$

Ainsi, de proche en proche, le coût  $C$  pour les paramètres  $\vec{\theta}$  du modèle devrait converger vers un minimum, comme on peut le représenter<sup>40</sup> dans le cas simplifié d'une fonction de 2 paramètres.



Or, d'après la règle de dérivation des composées (*chain rule*), on a

$$\frac{\partial C}{\partial w_{ji}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{ji}^{(L)}} \times \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \times \frac{\partial C}{\partial a_j^{(L)}} = \left( \sum_{i=1}^{n_{L-1}} a_i^{(L-1)} \right) \times \sigma'(z_j^{(L)}) \times \frac{\partial C}{\partial a_j^{(L)}}$$

Le calcul est **similaire en fonction des biais**. Sur la couche de sortie  $L_{\max}$ ,

$$\frac{\partial C}{\partial a_j^{(L_{\max})}} = 2 \sum_{k=1}^{n_{L_{\max}}} \left( a_k^{(L_{\max})} - y_k \right)$$

et sur les autres niveaux,

$$\frac{\partial C}{\partial a_j^{(L)}} = \sum_{k=1}^{n_L} \frac{\partial z_k^{(L+1)}}{\partial a_j^{(L)}} \times \frac{\partial a_k^{(L+1)}}{\partial z_k^{(L+1)}} \times \frac{\partial C}{\partial a_k^{(L+1)}} = \sum_{k=1}^{n_L} w_{ki}^{(L)} \times \sigma' z_k^{(L+1)} \times \frac{\partial C}{\partial a_k^{(L+1)}}$$

On peut donc propager les valeurs de **ces dérivées partielles**, du dernier rang  $L_{\max}$  vers le précédent et ainsi de suite en remontant de chaque couche vers celle d'avant. On est alors en mesure de calculer tous **les facteurs dont on a besoin**, grâce à cette phase dite de **rétro-propagation du gradient** (d'erreur). Il ne reste plus qu'à actualiser les valeurs des paramètres internes du modèle avec l'affectation (1).

Revenons à un point de vue plus global sur l'entraînement du modèle : on l'initialise avec des pondérations  $w_i$  et  $b_i$  aléatoires, puis on itère les étapes suivantes,

---

40. Illustration adaptée de  
<https://blog.paperspace.com/tag/series-gradient-descent-with-python/>

jusqu'à obtenir par exemple une précision suffisante sur le jeu de test, ou un coût assez faible<sup>41</sup> :

- On sélectionne aléatoirement une image du jeu d'entraînement (*c'est l'une des variantes de l'algorithme, voir plus bas*), qu'on soumet en entrée au réseau.
- On calcule les valeurs correspondantes des neurones de la première couche, puis de la suivante, etc. jusqu'à la dernière qui donne la supposée distribution des probabilités en sortie. C'est la phase de **propagation vers l'avant**.
- L'écart avec le résultat attendu est évalué avec la fonction de **coût**.
- On détermine comme on l'a détaillé ci-dessus son gradient, en remontant progressivement d'une couche vers la précédente, pour calculer les dérivées partielles du coût par rapport à chaque paramètre (phase de **rétro-propagation du gradient**<sup>42</sup> de l'erreur).
- Les pondérations synaptiques et les biais sont **mis à jour** avec (1).

On nomme *epoch* (terme généralement non traduit) un cycle d'apprentissage où toutes les données d'entraînement ont été traitées. En fait, plutôt que de procéder ainsi avec une soumission unique, on peut également « nourrir » le modèle avec un ensemble d'images (*batch*) avant d'actualiser les paramètres en se basant sur le coût moyen. On distingue plusieurs variantes de l'algorithme<sup>43</sup>, qu'on nomme généralement :

- **Stochastic Gradient Descent** SGD, qu'on vient de voir (*batch size* de 1).
- **Batch**, version d'origine de l'algorithme, où le lot est constitué de la totalité des données d'entraînement.
- **Mini-batch** pour les cas intermédiaires, où la *batch size* est strictement comprise entre 1 et la taille du *training dataset*. Il y a donc plusieurs mises à jour des paramètres du modèle (une après l'analyse de chacune des sorties associées à un lot), pour chaque *epoch*. Ce type de compromis permet de bénéficier à la fois de la stabilité de la convergence du *batch* et du gain en termes de calculs de la SGD (En pratique, l'acronyme SGD désigne parfois par abus cette version).

La quantité considérable de calculs nécessaires a longtemps été l'un des freins au développement de ces algorithmes. L'écriture matricielle est plus synthétique et favorise l'optimisation. Heureusement, beaucoup d'opérations sont parallélisables et plusieurs *frameworks* implémentent les opérations de manière efficace, en tirant notamment partie des architectures spécifiques des GPU et TPU.

L'outil proposé par Facebook Research et que nous avons utilisé, PyTorch, est arrivé plus tardivement que TensorFlow de Google par exemple. Mais son succès semble grandissant. Il offre un calcul transparent du gradient : il suffit de demander de « surveiller » les opérations à effectuer dans la phase directe (attribut `auto_grad = True` pour les paramètres à entraîner). Ensuite, leur ajustement est effectué automatiquement à la demande. Ce monitoring peut être levé à tout moment, notamment

41. Le premier critère est coûteux et sera en général évalué moins fréquemment ; un coût qui continue à décroître quand l'*accuracy* se dégrade sur le lot de test est généralement signe d'*overfitting*.

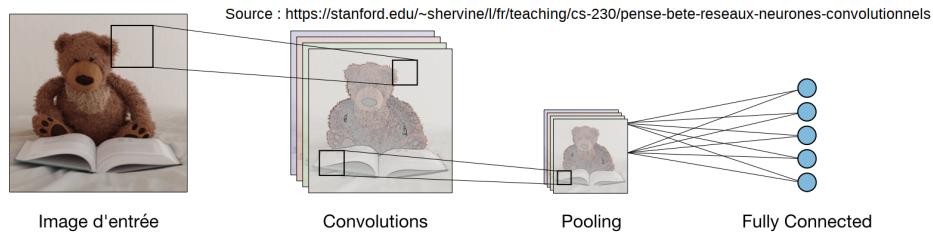
42. Si l'idée a été exposée par Seppo Linnainmaa dès 1970, elle s'est développée dans les années 80, notamment grâce aux travaux de David E. Rumelhart [RHW86], puis Yann Le Cun [LBD<sup>+</sup>89] qui l'a intégrée aux réseaux convolutifs, redonnant un nouvel élan au domaine.

43. Voir <https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size>

quand on calcule les sorties sur des données de test, pour éviter des calculs coûteux inutiles (exécution dans un bloc « `with torch.no_grad()` » par exemple). Optimiser les calculs grâce aux capacités d'un GPU est se limite à charger les variables sur ce support. La présence d'un tel équipement compatible avec CUDA est détectable automatiquement. Et on n'a pas à compiler un modèle à l'avance, on peut donc même l'adapter dynamiquement...

### 3.1.4 Réseaux de neurones convolutifs

Les réseaux « *fully connected* » qu'on vient d'analyser souffrent de l'explosion combinatoire du nombre de liaisons, notamment pour traiter des entrées de grande taille : le stockage des paramètres et la quantité de calculs à effectuer sont rapidement des facteurs limitants. Les CNN (alias ConvNets, *Convolutional Neural Network* ou réseau de neurones convolutifs) constituent une évolution du perceptron multicouche basique. Ils sont particulièrement adaptés au traitement des images<sup>44</sup>. Eux aussi sont inspirés par la biologie (compréhension du cortex visuel, dans les années 60 – 70). Le premier fût le *neocognitron*<sup>45</sup> de Kunihiko Fukushima en 1980. Le français Yann Le Cun<sup>46</sup> les a combinés avec profit avec la *backward propagation* en 1989 [LBD<sup>+</sup>89], pour reconnaître l'écriture manuscrite. Nous aurons l'occasion d'y revenir.



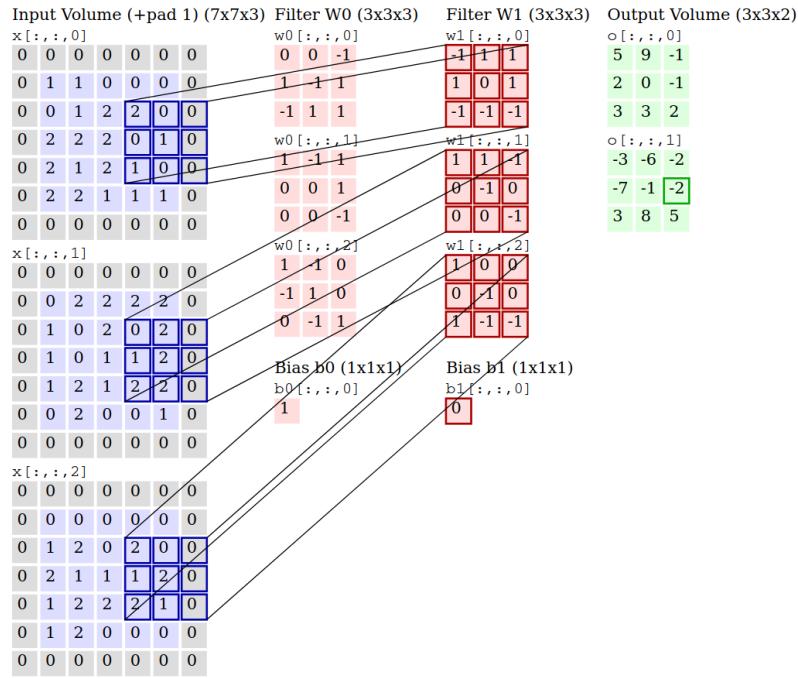
- Les **couches convolutives** (*convolutional layers*) donnent leur nom aux CNN. Elles ont pour but de « faire apparaître » des motifs exploitables dans les données de départ. On parle également de *feature map* ou *activation map*. Elles sont assimilables à une superposition de matrices (une par canal ou *channel*) : un torseur. Une image RGB par exemple peut être représentée sous cette forme avec 3 canaux. Un **filtre** (ou *kernel*) est également un petit torseur, qui balaye l'image ou les données en entrée, pour produire la couche suivante. À cet effet, un produit « terme à terme » est appliqué, puis les résultats sont additionnées. Si la profondeur est supérieure à 1 en entrée, on fait la somme des résultats obtenus sur chaque canal. Enfin, on peut ajouter un biais constant. Ces techniques sont celles utilisées dans des filtres graphiques classiques (déttection de contour, etc.) Mais ici, les coefficients sont des paramètres du réseau qui évoluent au même titre que les autres lors de l'apprentissage.

On peut paramétriser le pas pour le balayage du point d'application du filtre sur le torseur en entrée (*stride*, à 1 par défaut, suivant chaque dimension) ainsi que la taille des marges remplies de 0 ajoutées autour des matrices en entrée (*padding*, par défaut à 0).

44. Historique et détails sur [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network)

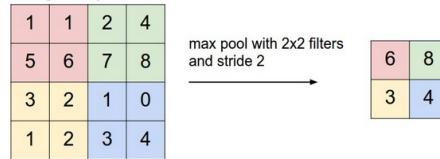
45. À lire sur [http://hal.univ-nantes.fr/docs/00/28/74/26/PDF/A\\_convolutional\\_neural\\_network\\_approach\\_for\\_objective\\_video\\_quality\\_assessment\\_completesfinal\\_manuscript.pdf](http://hal.univ-nantes.fr/docs/00/28/74/26/PDF/A_convolutional_neural_network_approach_for_objective_video_quality_assessment_completesfinal_manuscript.pdf) (document d'origine, que n'avons pas étudié).

46. Prix Turing 2019. [https://fr.wikipedia.org/wiki/Yann\\_Le\\_Cun](https://fr.wikipedia.org/wiki/Yann_Le_Cun)

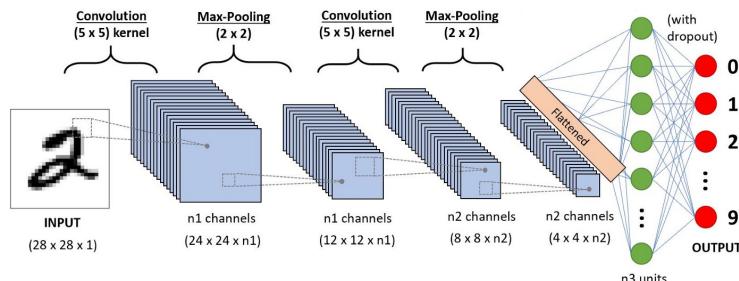


Si l'on souhaite une profondeur supérieure à 1 en sortie, on utilise un *kernel* à plusieurs canaux, chacun agissant comme un filtre indépendant qui fournit l'un des niveaux de profondeur en sortie. Il y en a deux notés,  $W_0$  et  $W_1$ , sur l'illustration précédente <sup>47</sup>.

- On peut ajouter des couches de *pooling* qui regroupent les valeurs de pixels voisins dans le but de sous-échantillonner et ainsi réduire la taille des données à traiter. Traditionnellement, on procède en remplaçant des groupes carrés (ou rectangulaires) de pixels de petites dimensions par leur maximum ou leur moyenne, en balayant là aussi les images ou données en entrée comme pour les convolutions. Là encore, *stride* et *padding* sont paramétrables. <sup>48</sup>



- On peut enchaîner plusieurs étapes de convolutions/*poolings*, puis on termine par les couches complètement connectées d'un perceptron pour assurer la classification.



47. Source : image animée sur <https://cs231n.github.io/convolutional-networks/>

48. Les deux illustrations suivantes proviennent respectivement de <https://pathmind.com/wiki/convolutional-network> et de <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

### 3.1.5 Implémentation avec PyTorch

Ce framework est disponible sous forme de package Python, il s'installe de manière standard via `pip3 install torch torchvision` ou équivalent. Il fournit les « briques » (structures et fonctions) conçues pour construire un modèle de réseau neuronal. Sa documentation <https://pytorch.org/docs/> est complète avec un moteur de recherche efficace. Il utilise massivement les tenseurs : ces tableaux  $n$ -dimensionnels implémentent ici la classe `Tensor` de manière analogue au type `ndarray` de Numpy.

Pour définir la structure d'un modèle, on définit une classe qui hérite de `nn.Module` (`nn` étant l'alias habituel pour `torch.nn`, à importer). On indique dans son constructeur les types et dimensions de chaque couche, attribut du modèle.

```
class MyConvNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 5) # in, out, kernel-sz
        self.fc1 = nn.Linear(16*5*5, 120) # in, out      [...]
```

On ajoute une méthode `forward` qui prend comme paramètre le *dataset* en entrée et renvoie le résultat de la phase de propagation directe, comme son nom l'indique. C'est donc là qu'on définit, dans l'ordre, les opérations à effectuer. On n'appelle pas directement cette fonction, mais on utilise à la place directement l'objet<sup>49</sup> (voir la phase d'entraînement plus bas).

```
# import torch.nn.functional as F
def forward(self, x):
    x = F.relu(self.conv1(x))
    x = F.max_pool2d(x, 2, 2) # kernel-sz, padding
    x = F.relu(self.fc1(x))   #           [...]
    return x
```

Une fois une instance implémentée, on définit la fonction de coût et l'objet qui prend en charge l'optimisation suivant la méthode choisie (ici `optimizer`).

```
# import torch.optim as optim
model = MyConvNet()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)
```

Le module `torchvision` fournit des modèles de CNN, pré-entraînés ou non, que l'on peut charger directement plutôt que définir sa classe à partir de zéro. Ils restent modifiables. Des fonctions de retouches d'images sont également proposées, qui permettent notamment d'enrichir un *dataset* trop petit, à l'aide de transformations aléatoires appliquées au jeu dont on dispose. Et surtout, il met à disposition des *datasets* classiques prêts à l'emploi<sup>50</sup>.

```
trainloader = torch.utils.data.DataLoader(
    datasets.MNIST(data_dir, train=True, download=True),
    batch_size=32, shuffle=True)
```

49. <https://pytorch.org/docs/stable/nn.html?highlight=parameter#torch.nn.Module.forward>

50. La liste des modèles disponibles (respectivement des *datasets* et des transformations) est accessible depuis <https://pytorch.org/docs/stable/torchvision/models.html> (respectivement depuis les pages `datasets.html` et `transforms.html`).

La phase d'entraînement exploite l'itérable implémentant `DataLoader` ainsi créé : remise à zéro des gradients, calcul des sorties avec les paramètres courants, calcul du coût associé, rétropropagation du gradient et mise à jour des paramètres.

```
model.train()          # Set model in training mode
losses = []
for inputs, labels in trainloader:
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    loss.backward()      # Compute gradient's backprop.
    optimizer.step()    # Update parameters
    losses.append(loss.item())
    # if ... : print(np.mean(losses))
```

Les tests sont conduits de manière analogue, tout en évitant les calculs de gradients et donc sans utilisation de l'`optimizer`.

```
model.eval()          # Set model in evaluation mode
with torch.no_grad(): # No backprop, no computation
    for inputs, labels in testloader:
        outputs = model(inputs)
        loss += criterion(outputs, labels).item()
loss /= len(testloader.dataset)
```

Notons enfin que l'optimisation sur GPU s'effectue simplement en chargeant sur la mémoire de ce dispositif le modèle et les données.

```
device_name = "cuda" if torch.cuda.is_available() else "cpu"
device = torch.device(device_name)
model = MyConvNet().to(device)
# Et dans la boucle d'entraînement par exemple
# inputs, labels = inputs.to(device), labels.to(device)
```

Les tutoriels du site officiel<sup>51</sup> complètent cette première présentation. Elle devrait néanmoins être suffisante pour comprendre la conversion d'un CNN en une version garante de la confidentialité différentielle (que nous noterons désormais **DP**, pour *Differential Private*). C'est l'objet de la partie suivante.

## 3.2 Garantie de confidentialité différentielle

### 3.2.1 Algorithme DP-SGD

L'introduction de bruit aléatoire pour assurer la confidentialité pourrait se faire à plusieurs niveaux. Il s'avère que pour ne pas trop nuire aux performances, c'est au gradient qu'il faut l'ajouter lors de l'apprentissage et non sur les images en entrée par exemple. Ainsi, on utilise l'algorithme dit de *Differentially Private Stochastic Gradient Descent* qu'on désignera par **DP-SGD**, obtenu en insérant les opérations suivantes dans la SGD classique, entre le calcul du gradient et son utilisation pour actualiser les paramètres, à savoir  $\vec{\theta} \leftarrow \vec{\theta} - \lambda \cdot \vec{\nabla}C(\vec{\theta})$ .

- On procède au **clipping de chaque gradient**, à savoir la majoration de sa norme par une valeur  $C > 0$ , grâce à un changement d'échelle si nécessaire. Le but de cette opération est de leur garantir une **sensibilité d'une valeur  $C$** .

---

51. [https://pytorch.org/tutorials/beginner/blitz/tensor\\_tutorial.html](https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html)

- On ajoute un bruit aléatoire qui suit une loi normale centrée et d'écart-type  $C\sigma$  à ces gradients modifiés (« mécanisme gaussien » de paramètre  $C\sigma$ ). La  $C$ -sensibilité garantit la  $(\alpha, \frac{\alpha}{2\sigma^2})$ -RDP du gradient (*cf.* 2.4.2). Rappons qu'on peut alors traduire cette formulation en  $(\varepsilon, \delta)$ -DP, plus explicite, en choisissant la valeur de  $\alpha$  qui donne le plus petit  $\varepsilon$ .

Ces deux phases sont prises en charge de manière transparente par le framework PyTorch-DP que nous étudierons. Présentons d'abord la base de donnée qui servira de support pour la présentation.

### 3.2.2 Exemple de MNIST

La *Modified National Institute of Standards and Technology database*<sup>52</sup> est un dataset constitué d'images de chiffres manuscrits de taille  $28 \times 28$  en niveaux de gris, associés à leur valeur numérique. On se compose de 60 000 images d'entraînement et 10 000 de test. Elle est devenue une référence en matière de reconnaissance d'images.

Il n'y a aucune donnée sensible à protéger ici, mais ce cadre est simple et éprouvé (*des biais présents dans les données initiales ont été corrigés, d'où le nom de la base*). Et l'absence d'enjeu réel n'entrave en rien l'exercice : la confidentialité différentielle revient ici à s'assurer que la présence ou non d'une image donnée dans la base utilisée lors de l'apprentissage est suffisamment peu influente sur le modèle qui en résulte pour ne pas être décelable en utilisant cet outil sous quelque forme que ce soit, même en ayant la connaissance complète du modèle et des hyper-paramètres utilisés pour l'apprentissage (qu'on nomme ainsi pour les distinguer des pondérations et biais du réseau neuronal : taille de *batch*, taux d'apprentissage, etc.)

Pour classifier ces images, nous choisissons le modèle exemple du dépôt GitHub de `facebookresearch/pytorch-dp`, dont nous étudierons ensuite la conversion en version DP : une couche convolutive à 16 canaux (*padding* de 3 et *stride* de 2), à la sortie de laquelle on applique ReLU puis un *max-pooling* de  $2 \times 2$ . Ensuite, on trouve une seconde couche de convolution à 32 canaux puis à nouveau les mêmes ReLU et *max-pooling*. Les données sont enfin réorganisées en un unique vecteur avant de passer par la partie « perceptron ». La première couche complètement connectée a 512 entrées. Elle est suivie de ReLU, puis d'une couche de taille 32, pour terminer par les 10 sorties associées à chacun des chiffres<sup>53</sup>.

L'expérience permet de ne pas choisir les dimensions complètement au hasard, mais il n'y a pas de règle systématique, l'approche reste empirique<sup>54</sup>. Il faut simplement prêter attention à la cohérence des dimensions d'une couche à la suivante, dans la partie convolutive. La figure 1 donne une vue d'ensemble du modèle et met ces contraintes en évidence, en illustrant le glissement du filtre de *pooling* et le nombre de canaux qui en découle impérativement sur la couche suivante.

Voyons à présent comment nous pouvons apporter une garantie de DP à notre modèle.

### 3.2.3 Implémentation avec PyTorch-DP

Abordons la transformation d'un réseau neuronal construit grâce à la version standard de PyTorch, afin qu'il garantisson la confidentialité différentielle des données

---

52. <http://yann.lecun.com/exdb/mnist/>

53. <https://github.com/facebookresearch/pytorch-dp/blob/master/examples/mnist.py> permet d'étudier les détails du constructeur de ce modèle.

54. D'ailleurs, la structure était différente quand nous avons commencé à étudier l'exemple du dépôt de `pytorch-dp`. Ce code est encore en version bêta et le site est très actif.

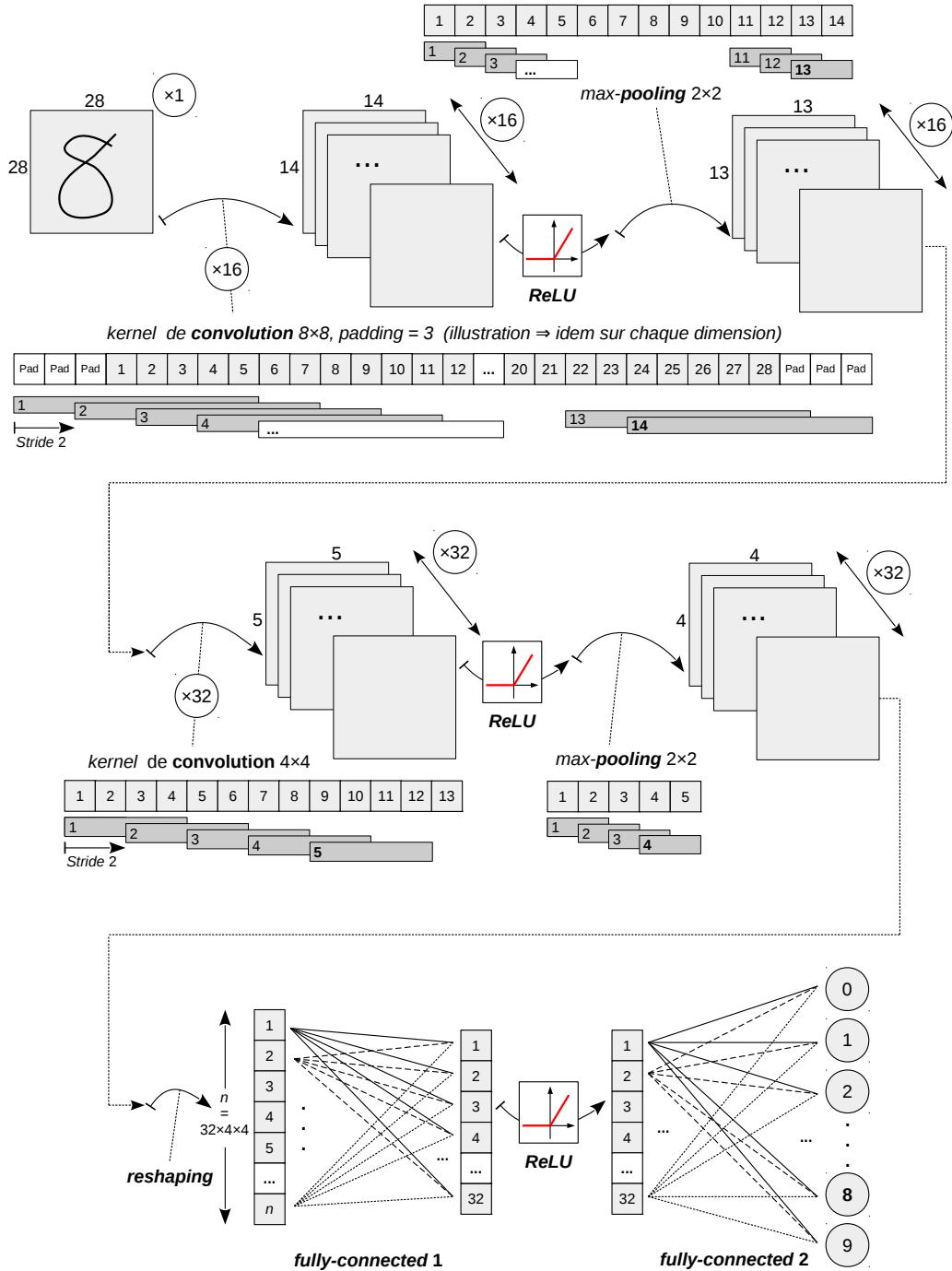


FIGURE 1 – Structure du CNN utilisé pour MNIST

d'entraînement. Celui présenté pour catégoriser les images de MNIST servira de support. Avec le framework PyTorch-DP, il suffit d'intervenir à deux niveaux :

- On modifie le modèle pour y intégrer la **DP-SGD** (*clipping* du gradient et ajout de bruit durant l’entraînement) en remplacement de la SGD classique.
  - Dans les cycles d’apprentissage, on intègre régulièrement un **calcul de la consommation du budget de confidentialité** qu’on affiche au fur et à mesure (c’est facultatif en fait, puisqu’on peut réaliser ce calcul *a priori* une

fois les hyper-paramètres connus).

L'installation de PyTorch-DP peut se faire à partir du dépôt précité :

```
git clone https://github.com/facebookresearch/pytorch-dp.git suivie de
pip3 install -e . met à disposition tous les outils nécessaires. De plus, l'exemple
de MNIST sera déjà intégré avec d'autres, en version DP, dans le dossier examples/.
```

Ce fichier `mnist.py` est utilisable en ligne de commande. Il définit la classe `SampleConvNet` du modèle et isole dans des fonctions `train()` et `test()` l'entraînement et la validation. Le reste du code, dans `main()`, se charge de récupérer le `dataset` (téléchargé automatiquement au besoin), d'instancier le CNN avant de lancer l'entraînement en fonction des paramètres passés au script (que `./mnist.py -h` permet de lister).

L'argument `--disable-dp` passé au script ramène à l'équivalent de la version PyTorch standard étudiée dans la partie précédente. Cela met en évidence le fait que la conversion du modèle à la DP se limite à l'ajout des deux parties incluses dans les blocs `if not args.disable_dp:`

- Dans `main()`, après avoir créé le CNN `model`, paramétré la SGD gérée par `optimizer` et avant de lancer la phase d'apprentissage :
  - On instancie un objet `privacy_engine` de la classe `PrivacyEngine`, à partir de `model` et des hyper-paramètres concernant la DP : taille de batch et du `dataset` d'entraînement, liste des coefficients  $\alpha$  utilisés pour la RDP, `noise multiplier` qui fixe le niveau de bruit ajouté, seuil de *clipping* du gradient.
  - Puis on appelle la méthode `attach()` qui va remplacer « à chaud » (*monkey patch*) l'optimizer associé à `model` par son équivalent DP.

```
model = SampleConvNet().to(device)
optimizer = optim.SGD(model.parameters(), lr=args.lr)

if not args.disable_dp:
    privacy_engine = PrivacyEngine(
        model,
        batch_size=args.batch_size,
        sample_size=len(train_loader.dataset),
        alphas=( [1 + x / 10.0 for x in range(1, 100)]
                  + list(range(12, 64)) ),
        noise_multiplier=args.sigma,
        max_grad_norm=args.max_per_sample_grad_norm)
    privacy_engine.attach(optimizer)
```

- Dans la fonction `train()`, il est possible de récupérer directement le budget de confidentialité consommé, en fonction de la valeur  $\delta$  ciblée, avec :

```
epsilon, best_alpha = \
    optimizer.privacy_engine.get_privacy_spent(args.delta)
```

Cette fonction renvoie  $\varepsilon$  en termes de  $(\varepsilon, \delta)$ -DP, ainsi que la valeur de  $\alpha$  qui a permis de déterminer la  $(\alpha, \varepsilon_{\text{rdp}})$ -RDP optimale associée.

Voici à présent quelques éléments de synthèse à propos du code de PyTorch-DP, que nous avons analysé pour déterminer les mécanismes et les justifications sous-jacentes (présentées précédemment, voir 3.2.1). Notons que cette démarche n'est pas indispensable pour ajouter la DP à un modèle, mais il n'était pas envisageable pour nous d'utiliser ce framework comme une « boîte noire » : en matière de confidentialité, le libre accès au code et aux propriétés qui le justifient est incontournable, d'autant plus que le projet est récent, tout comme les connaissances sur lesquelles il s'appuie. Nous avons d'ailleurs eu l'occasion de signaler un petit *bug*<sup>55</sup> et les publications supports doivent être abordées avec un regard critique car elles peuvent également comporter des erreurs<sup>56</sup>.

C'est le dossier `torchdp/` qui contient ce qui prend en charge la DP. Retenons principalement que :

- `privacy_engine.py` définit la classe `PrivacyEngine`, au cœur de la transformation, comme on l'a vu, avec sa méthode `attach()` qui permet d'insérer la gestion de la DP dans l'*optimizer* standard passé en paramètre. Ses méthodes `get_renyi_divergence()` et `get_privacy_spent(targetted_delta)` se chargent des calculs de budget associés en se basant sur des outils importés depuis `privacy_analysis.py`.
- `privacy_analysis.py`, reprend le code utilisé par Google TensorFlow Privacy. Il fournit essentiellement les deux fonctions appelées par les méthodes de `PrivacyEngine`, que sont `get_privacy_spent(orders, rdp, delta)` et `compute_rdp()`. Cette dernière prend comme paramètres le taux d'échantillonnage (ratio de la taille de *batch* sur celle des données d'entraînement), le niveau de bruit (noté précédemment  $C\sigma$ ), le nombre d'*epochs* et les coefficients  $\alpha$  à utiliser pour calculer la RDP.
- Les autres fonctions sont à usage interne, au service des deux précédentes (la figure 3 donne plus de détails).
- `per_sample_gradient_clip.py` et `autograd_grad_sample.py` sont en charge de l'ajout du bruit sur le gradient, notamment via la version DP de `step()` qui assure la mise à jour des paramètres lors de l'apprentissage.

La figure 2 détaille la cascade des appels liés à la transformation du modèle. La figure 3 présente celle des calculs du budget de confidentialité. Pour comprendre cette dernière, il est bon de revenir aux justifications (*cf.* 2.4.2).

Cette découverte de *deep learning* et de la prise en compte effective de la *differential privacy* dans ce cadre est terminée. Voyons à présent sous quelles formes concrètes ces notions ont pu se traduire dans le cadre de notre Master.

---

55. Une erreur d'arrondi sans grande conséquence, mais qui montre que des coquilles peuvent subsister. Sur PyTorch-DP <https://github.com/facebookresearch/pytorch-dp/issues/19> elle a été corrigée, mais pas encore du côté de TensorFlow Privacy <https://github.com/tensorflow/privacy/issues/103> qui en donnait la version d'origine, à l'heure où nous écrivons ces lignes.

56. Là aussi, nous avons découvert qu'une inégalité était à l'envers dans l'une des principales propriétés d'un article d'Ilya Mironov (théorème 4 [MTZ19]) et les corollaires suivants étaient invalides avec la forme fournie. Cela n'avait pas encore été repéré dans cette publication de 2017 et l'auteur a confirmé notre analyse. Fort heureusement, c'est la valeur correspondant au cas d'égalité qui est utilisée dans les implémentations de TensorFlow Privacy et PyTorch-DP !



FIGURE 2 – Cascade des appels pour la transformation du modèle en version DP.

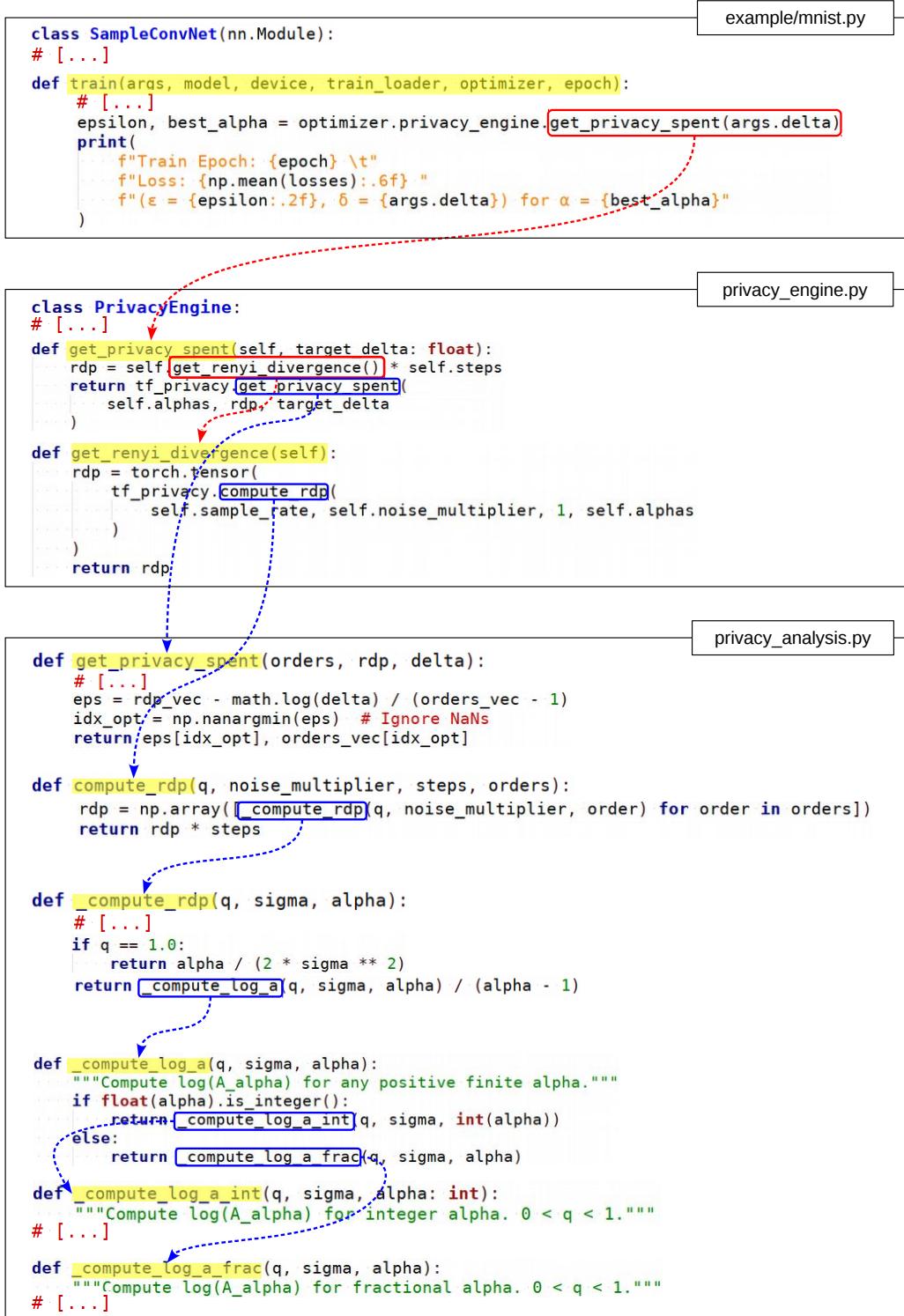


FIGURE 3 – Cascade des appels concernant le budget de confidentialité.

## 4 Expérimentations et productions personnelles

Dans toute cette partie, nous nous travaillerons exclusivement sur les réseaux neuronaux convolutifs (voire de simples perceptrons multicouches), destinés à classifier des données, en l'occurrence des images.

Pour mener à bien notre travail, nous avons bénéficié d'un accès distant sans restriction aux ordinateurs de l'institut FEMTO-ST. Sans ce support, il n'aurait guère été envisageable d'aller bien au delà du seul travail présenté dans le paragraphe suivant. La grande disponibilité de ce matériel nous a permis un nombre d'expérimentations conséquent et nous en sommes particulièrement reconnaissant à l'équipe AND du département DISC, qui nous l'a mis à disposition.

### 4.1 Calcul *a priori* du budget de confidentialité

La phase d'apprentissage d'un modèle DP inclut en principe des calculs réguliers de la consommation du budget de confidentialité au stade courant, celle-ci étant affichée au fur et à mesure. Mais l'entraînement d'un réseau de convolution est coûteuse en temps et en calculs. Pour les différents choix des hyper-paramètres envisageables, il est donc intéressant de prévoir dès le départ quel est le coût associé, en termes de confidentialité différentielle. À cette fin, nous avons donc écrit un script Python, dans le cadre du *Sampled Gaussian Mechanism* autrement dit celui de la descente de gradient DP-SGD étudiée au 3.2.1.

#### 4.1.1 Écriture d'un script dédié

Rappelons que PyTorch est un *framework* plus récent que ses principales alternatives, comme TensorFlow, l'une des références incontournables en la matière (on les doit respectivement à Facebook Research et Google). Cependant, au niveau de leurs déclinaisons respectueuses de la confidentialité, PyTorch-DP et TensorFlow Privacy, l'écart est moins évident. Leur développement est récent et reste très actif pour les deux plateformes.

`pypi.org` est le dépôt source par défaut pour les modules Python installés avec pip. À ce jour on y trouve une version 0.3.0 qui a moins d'un an pour le module `tensorflow-privacy`, mais dont la page de présentation est encore vide. De son côté, `pytorch-dp` offre une documentation, mais il n'est proposé qu'en version 0.1.0b1. Cela n'a rien de surprenant puisqu'il n'est disponible que depuis moins de deux mois. Heureusement, nous avions eu accès plus tôt à cet outil, depuis son dépôt GitHub.

Les deux étant publiés sous *Apache License 2.0*, PyTorch-DP a pu reprendre les fonctions de bas niveau de l'environnement TensorFlow Privacy en charge du calcul de budget, que nous préciserons plus loin. Ce dernier dispose également d'un script qui permet d'accéder à ces informations indépendamment de l'entraînement ou même de l'instanciation d'un réseau neuronal. Nous nous en sommes donc fortement inspiré pour construire son équivalent pour PyTorch-DP. Notre proposition a été acceptée et intégrée au projet officiel, pour notre plus grand plaisir. Cette adaptation a également été l'occasion de repérer et corriger un bug qui engendrait une légère erreur dans la valeur renvoyée, dans le cas où la taille de `batch` n'est pas un diviseur de celle de la base d'entraînement. Cela illustre l'un des intérêts de la mise en commun de code sous licence libre, avec un exemple, certes modeste, de « retour sur investissement »<sup>57</sup>.

---

57. Quoi que, comme nous l'avions déjà indiqué, l'erreur a été corrigée sur PyTorch-DP <https://github.com/facebookresearch/pytorch-dp/issues/19>, mais pas encore à ce jour pour TensorFlow Privacy <https://github.com/tensorflow/privacy/issues/103> !

Dans la version de TensorFlow Privacy, c'est `flags.FLAG` du module `absl` de Google qui est utilisé pour l'analyse des paramètres de la CLI. Nous l'avons remplacé par `argparse`. Cela ne change rien sur le fond, mais vu que c'est lui qui était utilisé dans les exemples de PyTorch-DP, ce choix paraissait plus cohérent.

Par ailleurs, les fonctions de haut niveau renvoyant au final le budget consommé n'étaient pas encore présentes sur PyTorch-DP et ne pouvaient donc pas être importées comme dans TensorFlow Privacy. Nous les avons intégrées au script lui-même. Elles s'appuient sur les fonctions incluses dans `privacy_analysis.py`, utilisées comme dans un réseau réellement instancié.

Il s'agit principalement de `compute_dp_sgd_privacy()`, qui prend comme paramètres les tailles du *dataset* d'entraînement et du *mini-batch*, le « facteur de bruit » `noise_multiplier` qui permet de doser la perturbation ajoutée, le nombre d'*epochs* et la valeur ciblée pour  $\delta$ . De manière facultative, on peut opter pour une autre liste `alphas` de coefficients pour le calcul de la RDP, dans le cas où celle qui est choisie par défaut ne conviendrait pas. Il est également possible d'éviter les affichages des informations sur la sortie standard (avec `printed=False` en paramètre).

La fonction `apply_dp_sgd_analysis()` utilisée par la précédente peut aussi être appelée directement, à condition d'expliquer en argument le `sample_rate`, quotient des tailles de *mini-batch* et de *dataset*, ainsi que le nombre `steps` d'itérations du mécanisme gaussien. Les deux renvoient un `tuple`, formé du coefficient  $\epsilon$  le plus bas qui a pu être déterminé en termes de  $(\epsilon, \delta)$ -DP, ainsi que du  $\alpha$  correspondant au niveau de l' $(\alpha, \epsilon_{\text{rdp}})$ -RDP.

Il suffit donc d'importer `compute_dp_sgd_privacy` pour calculer un budget avec un simple appel à l'une de ces fonctions, en Python. De manière alternative, on peut utiliser ce fichier comme un script `shell` (CLI). Comme souvent, un appel avec le paramètre `-h` permet de lister toutes les options utilisables.

```
$ ./compute_dp_sgd_privacy.py -s 60000 -b 64 -n 1.0 -e 15
=====
* Script was called with arguments:
  -s = --dataset-size = 60000
  -b = --batch-size = 64
  -n = --noise_multiplier = 1.0
  -e = --epochs = 15
  -d = --delta = 1e-05
  -a = --alphas = [1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0,
2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4,
3.5, 3.6, 3.7, 3.8, 3.9, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8,
4.9, 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0, 6.1, 6.2,
6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 7.0, 7.1, 7.2, 7.3, 7.4, 7.5, 7.6,
7.7, 7.8, 7.9, 8.0, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6, 8.7, 8.8, 8.9, 9.0,
9.1, 9.2, 9.3, 9.4, 9.5, 9.6, 9.7, 9.8, 9.9, 10.0, 10.1, 10.2, 10.3,
10.4, 10.5, 10.6, 10.7, 10.8, 10.9, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
54, 55, 56, 57, 58, 59, 60, 61, 62, 63]
=====
* Result is:
  DP-SGD with
    sampling rate = 0.107% and
    noise_multiplier = 1.0
    iterated over 14070 steps
  satisfies differential privacy with
    ε = 1.17 and
    δ = 1e-05.
  The optimal α is 13.0.
=====
```

Nous avons illustré la cascade des appels de fonctions dans la figure 4 et le code source complet est accessible sur [https://github.com/facebookresearch/pytorch-dp/blob/master/torchdp/scripts/compute\\_dp\\_sgd\\_privacy.py](https://github.com/facebookresearch/pytorch-dp/blob/master/torchdp/scripts/compute_dp_sgd_privacy.py).

#### 4.1.2 Documentation (carnet Jupyter)

En réalité, c'est l'étude du code de l'exemple à propos de MNIST inclus dans PyTorch-DP, en vue d'en comprendre les justifications, qui nous a amené à écrire le script précédent. Il nous a alors paru pertinent de partager le résultat de cette

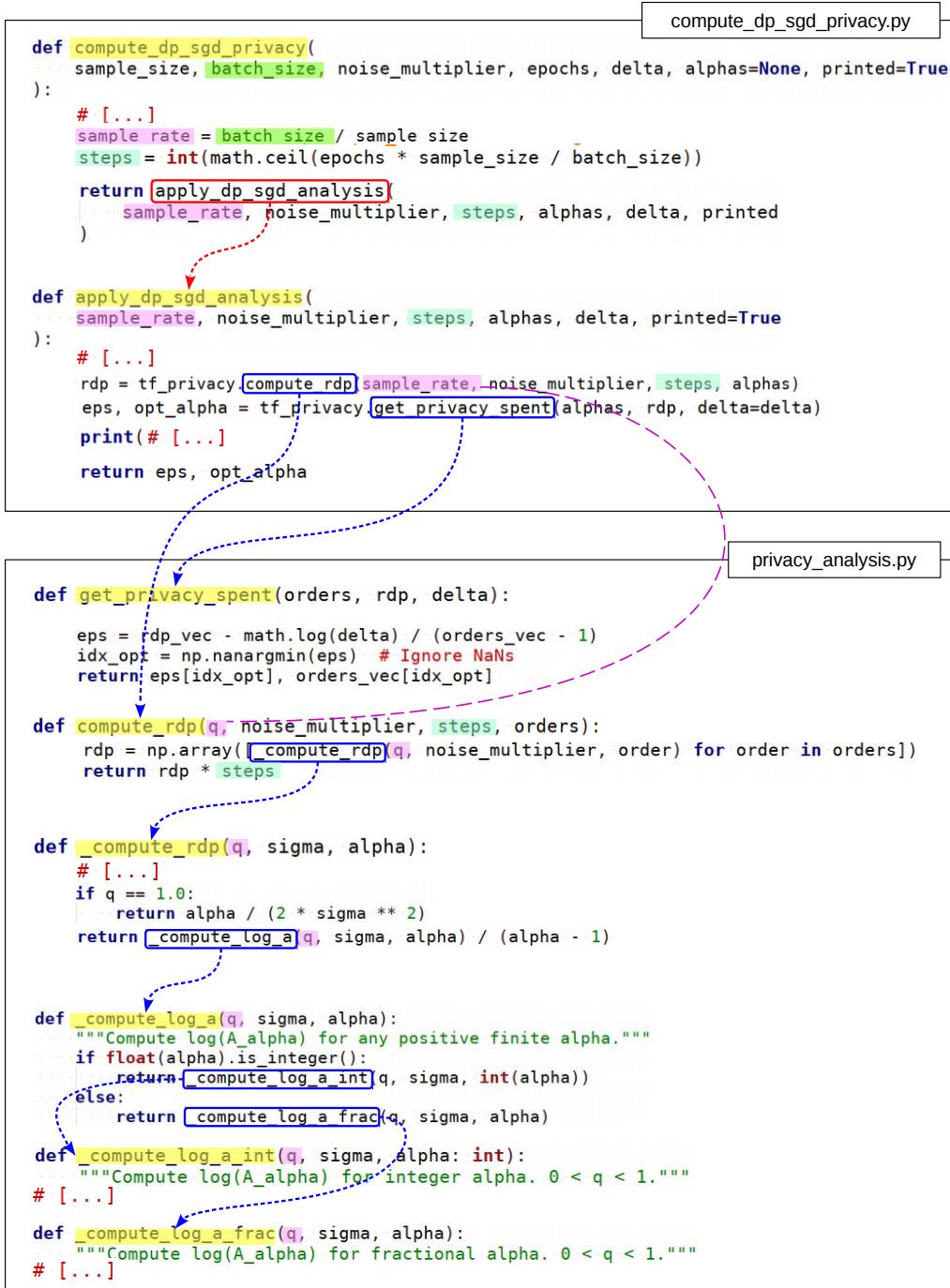
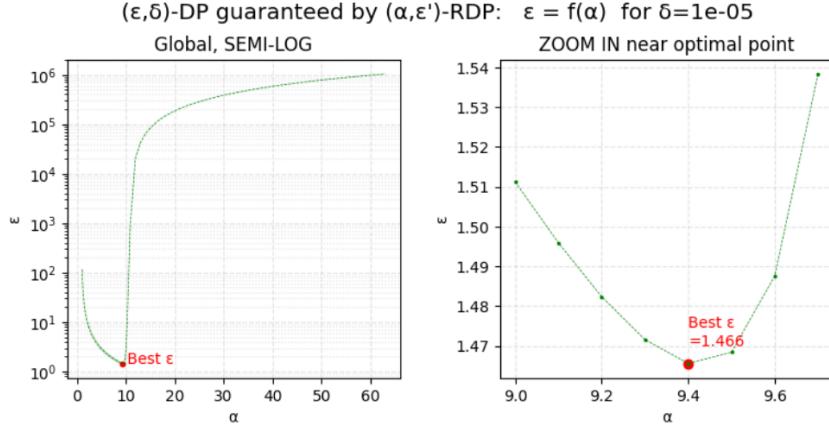


FIGURE 4 – Cascade des appels pour le calcul de budget *a priori*.

exploration, à destination des utilisateurs du *framework*. Étant donné le public ciblé, nous avons choisi de le présenter en anglais, sous la forme d'un *Jupyter notebook*. Ce format permet en effet d'alterner descriptions et code, tout en laissant à l'utilisateur la possibilité d'interagir. L'intéressé peut expérimenter, adapter le code à loisir, sans installation. En effet, suivre un simple lien peut suffire, pour peu que l'on s'appuie une instance de Jupyter accessible sur le web (on peut utiliser un service comme [mybinder.org](https://mybinder.org) si l'on ne dispose pas de son propre serveur).

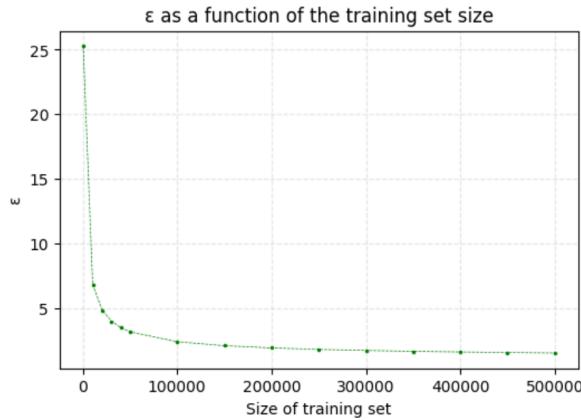
Ce « carnet » débute par une courte démonstration d'appels aux fonctions de cal-

culs de budget de DP, puis de l'utilisation comme script. Nous avons ensuite rappelé la manière dont  $(\alpha, \varepsilon_{\text{rdp}})$ -RDP et  $(\varepsilon, \delta)$ -DP étaient utilisées de concert par PyTorch-DP et pointé les articles qui justifient et expliquent les calculs correspondants. Cela a été illustré avec la représentation suivante de  $\varepsilon$  en fonction de  $\alpha$ . Son allure permet notamment de percevoir pourquoi les valeurs d' $\alpha$  sont de plus en plus espacées quand elles augmentent, dans les listes utilisées par PyTorch-DP pour discréteriser cette courbe.



Enfin, nous avons intégré quelques graphiques illustrant la dépendance d' $\varepsilon$  aux différentes données numériques qui interviennent. C'est dans cette partie que l'interactivité prend tout son sens, car l'utilisateur peut adapter l'observation à son propre contexte.

- **Quantité des données d'entraînement** : bien qu'en général on ait peu le choix à ce sujet, il est bon d'en comprendre l'importance. Nous observons qu'un effectif trop faible rend difficile l'obtention d'une confidentialité suffisante. Cela met en évidence l'intérêt de la création artificielle des données « de synthèse ». À cet effet, nous pouvons par exemple faire pivoter aléatoirement des images fournies, les décentrer, les flouter ou en jouer sur tout autre paramètre pour créer de nouvelles entités.
- À l'inverse, au delà d'un certain seuil, le tracé nous montre que le nombre d'entrées aura à lui seule une influence modeste sur le coefficient  $\varepsilon$ .

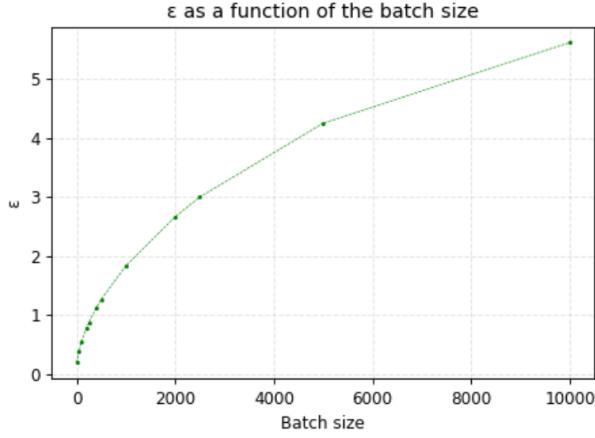


En fait, l'idée était principalement de s'intéresser aux différents hyper-paramètres, qui servent de « boutons de réglage » de la confidentialité.

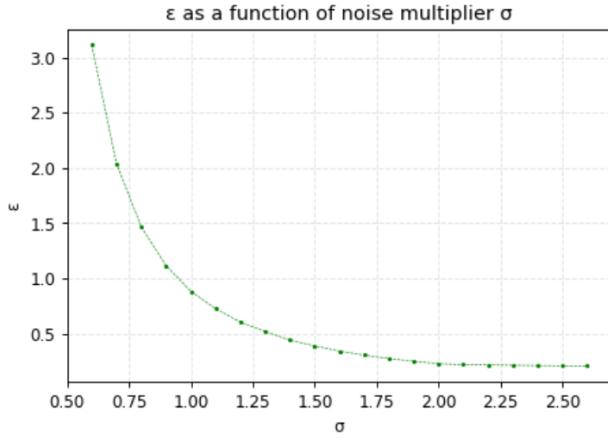
- **Taille des mini-batches** : plus petits, ils ajoutent de l'aléa, favorable à la confidentialité. Mais il ne faut pas oublier qu'une plus grande taille semble

favoriser la convergence. Au final, il convient de rester prudent quant à l'interprétation de ce paramètre, nous y reviendrons.

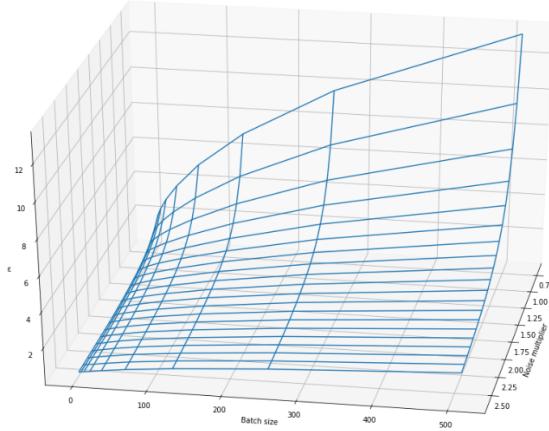
Indépendamment, pour éviter des séries incomplètes sur la dernière itération de chaque *epoch*, on privilégiera une valeur qui divise le nombre d'entrées (ce qui est le cas pour les points du graphique suivant).



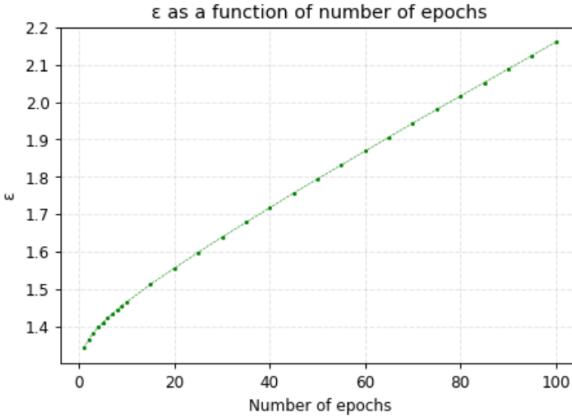
- **Quantité de bruit ajouté** : nous pouvons voir que l'augmenter est d'abord très efficace pour les petites valeurs, mais que son effet s'amenuise ensuite, alors que la précision du modèle risque de pâtir sérieusement d'un niveau trop élevé.



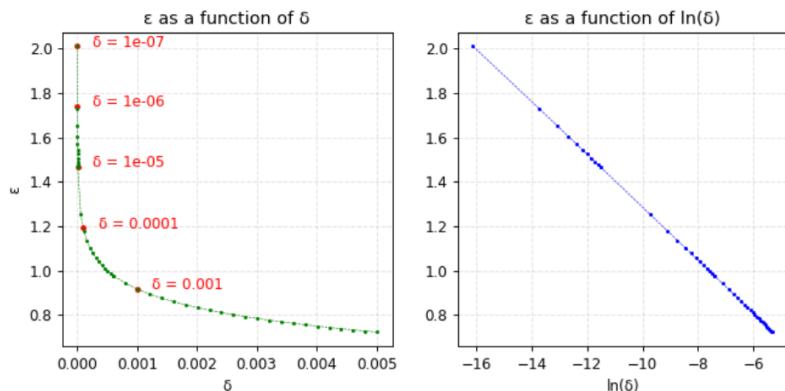
On peut également s'intéresser à l'importance relative du bruit et de la taille de batch, car c'est souvent ce compromis qu'il faudra gérer, notre pratique a pu nous le confirmer.



- **Nombre d'epochs** : dès le premier cycle,  $\varepsilon$  a déjà une valeur non négligeable qui dépend des autres hyper-paramètres. La vitesse de dégradation de la confidentialité est la plus élevée au début. Elle diminue ensuite rapidement, et son évolution devient très linéaire.



- **delta** : bien entendu, son choix est avant tout lié au risque que l'on accepte de prendre quant à la divulgation d'informations sensibles, suivant les données utilisées, ainsi qu'à leur nombre (une valeur supérieure à l'inverse de l'effectif du *training dataset* rend la protection caduque). Les représentations suivantes montrent cependant que prendre une valeur trop basse « par sécurité » n'est pas sans conséquence sur le coût  $\varepsilon$ .



#### 4.1.3 Justifications des calculs sous-jacents

Ce *notebook* est en cours d'intégration, lui aussi, dans le projet PyTorch-DP. Comme il est disponible en ligne<sup>58</sup>, nous n'allons pas détailler davantage son contenu ici. Mais nous allons approfondir un point central et peu évident parmi ce que nous avons mis au jour, « sous le capot » de PyTorch-DP en quelque sorte. Ces détails ne sont pas donnés dans le carnet. Nous nous intéressons donc ici, dans le cas de la DP-SGD, au seul calcul du coefficient  $\varepsilon_{\text{rdp}}$  de la RDP. Rappelons qu'il est ensuite possible de le traduire en  $(\varepsilon, \delta)$ -DP.

Supposons que le gradient a comme sensibilité  $C$ , ce qu'un *clipping* à ce seuil assure, en pratique. Le bruit ajouté suit une loi normale centrée d'écart-type  $C\sigma$ .

<sup>58</sup>. À terme, [https://github.com/facebookresearch/pytorch-dp/docs/DP\\_Computation\\_in\\_SGM.ipynb](https://github.com/facebookresearch/pytorch-dp/docs/DP_Computation_in_SGM.ipynb) devrait pointer sur ce *notebook*. En attendant, vous pourrez le consulter et l'exécuter depuis <https://github.com/jmg-74/exam/docs/>.

- Pour le cas particulier sans échantillonnage, où le *batch* est constitué du lot d'entraînement complet, on reprendra le résultat déjà énoncé pour le mécanisme gaussien :

$$\varepsilon_{\text{rdp}} = \frac{\alpha}{2\sigma^2}$$

- Si on travaille par *mini-batches* correspondant à un taux d'échantillonnage de  $q$  (ratio de la taille de ces paquets sur celle des données d'entraînement), alors Ilya Mironov *et al.* montrent<sup>59</sup> qu'on peut prendre

$$\varepsilon_{\text{rdp}} = \frac{1}{\alpha - 1} \ln A_\alpha$$

où

$$A_\alpha = \mathbb{E}_{z \sim \mu_0} \left[ \left( \frac{\mu(z)}{\mu_0(z)} \right)^\alpha \right] = \int_{-\infty}^{+\infty} \left( \frac{\mu(z)}{\mu_0(z)} \right)^\alpha \mu_0(z) dz$$

en notant  $\mu_0$  la densité de la loi normale centrée d'écart-type  $\sigma$  notée  $\mathcal{N}(0, \sigma^2)$ ,  $\mu_1$  celle de  $\mathcal{N}(1, \sigma^2)$  centrée en 1 et  $\mu = (1 - q)\mu_0 + q\mu_1$  soit

$$A_\alpha = \mathbb{E}_{z \sim \mu_0} \left[ \left( (1 - q) + q \frac{\mu_1(z)}{\mu_0(z)} \right)^\alpha \right] = \int_{-\infty}^{+\infty} \left( (1 - q) + q \frac{\mu_1(z)}{\mu_0(z)} \right)^\alpha \mu_0(z) dz$$

Cela ramène donc au calcul de ce nombre  $A_\alpha$  en fonction de  $q$  et  $\alpha$ . Deux cas sont alors envisagés :

- Quand  $\alpha$  est entier : en utilisant la formule du binôme,

$$\left( \frac{\mu(z)}{\mu_0(z)} \right)^\alpha = \sum_{k=0}^{\alpha} (1 - q)^{\alpha-k} q^k \left( \frac{\mu_1(z)}{\mu_0(z)} \right)^k$$

et on montre en revenant à la définition de la loi normale que

$$\mathbb{E}_{z \sim \mu_0} \left[ \left( \frac{\mu_1(z)}{\mu_0(z)} \right)^k \right] = \exp \left( \frac{k^2 - k}{2\sigma^2} \right)$$

d'où finalement

$$A_\alpha = \sum_{k=0}^{\alpha} (1 - q)^{\alpha-k} q^k \exp \left( \frac{k^2 - k}{2\sigma^2} \right)$$

qui permet un calcul direct par itérations.

- Pour un  $\alpha$  non entier, on utilise

– les coefficients binomiaux généralisés<sup>60</sup>

$$\binom{\alpha}{k} = \frac{\alpha(\alpha - 1) \cdots (\alpha - k + 1)}{k!}$$

et la série associée

$$(x + y)^\alpha = \sum_{k=0}^{+\infty} \binom{\alpha}{k} x^{\alpha-k} y^k \quad \text{convergente quand } |x| < |y|$$

59. Voir [MTZ19] sur lequel s'appuient également tous les calculs suivants du paragraphe.

60. Voir [https://fr.wikipedia.org/wiki/Formule\\_du\\_binôme\\_généralisé](https://fr.wikipedia.org/wiki/Formule_du_binôme_généralisé)

- Un découpage astucieux pour se ramener à deux séries convergentes : on choisit  $z_1 = \frac{1}{2} + \sigma^2 \ln\left(\frac{1-q}{q}\right)$ .

Ainsi, pour  $z < z_1$ , on a  $1-q < q \frac{\mu_1(z)}{\mu_0(z)}$ , d'où en prenant  $x = 1-q$  et  $y = q \frac{\mu_1(z)}{\mu_0(z)}$  dans la formule précédente,

$$\left(\frac{\mu(z)}{\mu_0(z)}\right)^\alpha = \left((1-q) + q \frac{\mu_1(z)}{\mu_0(z)}\right)^\alpha = \sum_{k=0}^{+\infty} \binom{\alpha}{k} (1-q)^{\alpha-k} q^k \left(\frac{\mu_1(z)}{\mu_0(z)}\right)^k$$

Et pour  $z > z_1$ , un raisonnement analogue amène à :

$$\left(\frac{\mu(z)}{\mu_0(z)}\right)^\alpha = \sum_{k=0}^{+\infty} \binom{\alpha}{k} (1-q)^k q^{\alpha-k} \left(\frac{\mu_1(z)}{\mu_0(z)}\right)^{\alpha-k}$$

Il ne reste alors qu'à séparer l'intégrale généralisée en deux parties et sommer les termes sur  $k$  jusqu'à obtenir une approximation suffisante de la limite. En effet, on montre que

$$\int_{-\infty}^{z_1} \left(\frac{\mu_1(z)}{\mu_0(z)}\right)^k \mu_0(z) dz = \frac{1}{2} \exp\left(\frac{k^2 - k}{2\sigma^2}\right) \operatorname{erfc}\left(\frac{k - z_1}{\sigma\sqrt{2}}\right)$$

et

$$\int_{z_1}^{+\infty} \left(\frac{\mu_1(z)}{\mu_0(z)}\right)^k \mu_0(z) dz = \frac{1}{2} \exp\left(\frac{k^2 - k}{2\sigma^2}\right) \operatorname{erfc}\left(\frac{z_1 - k}{\sigma\sqrt{2}}\right)$$

sachant que la fonction d'erreur complémentaire  $\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{+\infty} e^{-t^2} dt$  est calculée en se basant sur le module `scipy`.

À défaut de passionner tout informaticien, ces détails ont pour objectif de permettre à chacun de comprendre plus facilement le code source des fonctions utilisées en interne dans le fichier `privacy_analysis.py`. À cet effet, ajoutons encore que les calculs sont effectués « en espace logarithmique » (en travaillant sur les logarithmes des valeurs pour les étapes intermédiaires), pour éviter les pertes de précision dues aux limitations de la représentation des flottants.

Revenons à présent à du concret. Voici une présentation de notre adaptation à la DP de réseaux neuronaux.

## 4.2 Conversion à la DP de réseaux de classification

### 4.2.1 Environnement de travail

Nous avons eu accès à un **cluster du département DISC de FEMTO-ST**, qui nous permettait de lancer les entraînements de nos modèles sur des machines bien équipées (double Intel(R) Xeon(R) E5-2623 v4 @ 2.60GHz, i7-8700 @ 3.20GHz ou i9-9900K @ 3.60GHz avec 64 Go de RAM), disposant surtout de cartes graphiques NVIDIA GM200 GeForce GTX TITAN X à 12 Go de mémoire vive, embarquant 3 072 coeurs CUDA. L'accès était effectué de manière classique via `ssh`, d'abord sur le nœud d'identification, puis depuis là vers celui choisi pour effectuer les calculs.

Nous avons malgré tout atteint les limites de ces configurations comme nous le verrons, essentiellement au niveau de la mémoire vive du GPU. De nouvelles tentatives ont alors été effectuées sur le **calculateur du Mésocentre de calcul de Franche-Comté** où 90 Go de RAM nous étaient alloués (avec 16 Go sur GPU). La démarche était un peu différente. Sur une telle infrastructure régionale, assez

impressionnante<sup>61</sup>, dont les moyens peuvent d'ailleurs être loués par les industriels, nous n'exécutons pas directement nos programmes Python. Ils sont soumis via un langage spécifique, SGE<sup>62</sup> : une requête est envoyée par la commande `qsub` via un script écrit dans ce langage ou directement depuis la ligne de commande. Un « *job* » est alors créé et mis en file d'attente. Il sera exécuté dès que possible en fonction des disponibilités, `qstat` permet de suivre la situation, `qdel` de demander l'annulation d'un *job* en attente ou en phase d'exécution. En fin de cycle, la sortie est redirigée dans un fichier indiqué dans la commande initiale, qui nous est renvoyé (nous n'avons pas eu à travailler de manière interactive).

Évidemment, la mise à disposition de tels moyens a été déterminante, d'autant plus que nous ne disposions pas de processeur graphique ou spécialisé du type TPU.

En effet, les capacités des GPU sont particulièrement utiles à PyTorch(-DP) pour accélérer les traitements. Et ce framework permet de bénéficier de l'environnement CUDA presque automatiquement : l'une des méthodes `.cuda()`, ou `.to(my_dev)` pour `my_dev = torch.device('cuda')`, suffit à charger un modèle ou les données d'un *dataset* sur le GPU. Il est également possible de l'indiquer en paramètre, à la création d'un torseur : `torch.Tensor([0., 1.], device=my_dev)`. Si plusieurs processeurs graphiques sont à disposition simultanément, il suffit de remplacer `'cuda'` par `'cuda:0'` ou `'cuda:1'`, etc. pour préciser son choix.

L'exploitation explicite du parallélisme est également possible, en utilisant `torch.nn.DataParallel`, mais nous n'avons pas exploré ce domaine.

#### 4.2.2 Expérimentations

Notre objectif était d'abord de valider notre capacité à convertir un réseau neuronal de classification à la confidentialité différentielle. Par ailleurs, nous voulions étudier l'impact de cette mutation sur la précision de ce modèle, notamment suivant les hyper-paramètres choisis. La performance du réseau convolutif de départ était donc secondaire, pour peu qu'elle reste décente.

Nous avons cherché un exemple pas encore traité avec PyTorch-DP. Cette quête nous a conduit à la base de données que nous nommerons « *102 flowers* », proposée par l'*University of Oxford Visual Geometry Group*. Il s'agit à nouveau de classifier des images mais les différences avec MNIST suffisent à lui donner de l'intérêt : elles sont en couleurs RGB, avec une définition de  $224 \times 224$ , il y a 102 catégories différentes et on ne dispose que de 40 à 258 clichés pour chacune, sur les 6 552 entrées d'entraînement.

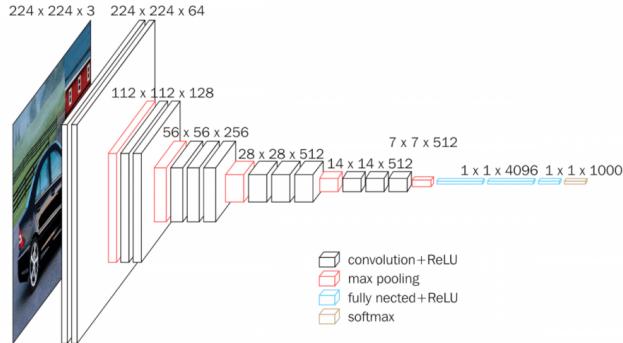
Pour construire le réseau neuronal, nous nous sommes basé sur l'architecture VGG16 ([SZ15]<sup>63</sup>) qui a déjà montré son efficacité dans ce domaine et qui restait dans le cadre fixé (réseau neuronal convolutif). Les images d'origine ne sont pas en  $224 \times 224$ , mais sont ramenées à ce format lors de la préparation du *dataset* pour être adaptées à VGG16.

---

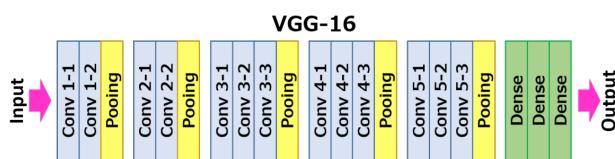
61. Pour les amateurs de chiffres : 141 nœuds, 2 292 coeurs, 9,27 To de mémoire, 74,26 Tflops sur CPU et 66,48 Tflops sur GPU...

62. Pour *Sun Grid Engine*, même si Oracle, puis Univa ont pris le relai, voir [https://en.wikipedia.org/wiki/Oracle\\_Grid\\_Engine\Univa\Oracle](https://en.wikipedia.org/wiki/Oracle_Grid_Engine\Univa\Oracle). Le classique `man` donne l'utilisation des commandes disponibles, `qsub`, `qstat` et `qdel` étant suffisantes en ce qui nous concerne. On trouve un bon résumé sur [http://gridscheduler.sourceforge.net/howto/basic\\_usage.html](http://gridscheduler.sourceforge.net/howto/basic_usage.html) et cette autre page détaillent simplement les utilisations directes ou par script.

63. Une présentation est donnée sur <https://neurohive.io/en/popular-networks/vgg16/> dont les illustrations concernant VGG16 sont tirées.



Ce qui distingue clairement VGG16 de la structure choisie dans l'exemple du MNIST est la profondeur de la partie convolutive, basée sur des noyaux de petites tailles, en amont du perceptron multicouche.



D'ailleurs, pour gagner du temps d'apprentissage, nous sommes parti d'un modèle pré-entraîné (sur des images librement accessibles, donc pas à protéger, tout en supposant arbitrairement le contraire concernant nos images à classifier). À nouveau, le confort offert par l'environnement PyTorch est appréciable, puisque ce réseau est proposé par le module `torchvision.models`, entraîné ou non. C'est le cas pour un grand nombre d'architectures classiques publiées. Nous avons « figé » la partie convolutive, pour n'entraîner que les couches denses du perceptron. Ce choix a été validé par une précision (*accuracy*) qui devenait rapidement correcte, en version non-DP.

Cette approche avait un autre avantage potentiel : l'ajout de bruit n'intervenant que sur la dernière partie, on pouvait espérer une moindre dégradation du modèle en passant à la DP. Pour y parvenir, avec PyTorch il suffit de modifier un attribut des couches concernées, pour indiquer qu'elles ne doivent pas être modifiées en fonction du gradient de l'erreur :

```
# Freeze existing model parameters for training
for param in model.parameters():
    param.requires_grad = False
```

La pratique nous a montré qu'un compromis acceptable pouvait s'avérer difficile voire impossible à trouver lors du passage en version DP. C'est pourquoi, afin d'analyser la situation en disposant d'éléments de comparaison, nous avons également travaillé sur le CIFAR10<sup>64</sup>. Il s'agit d'une base de 60 000 images  $32 \times 32 \times 3$  dont 50 000 pour l'entraînement, à classer dans 10 catégories (avion, voiture, camion, chat, chien, etc.). L'effectif de chacun des groupes est identique. La principale différence avec MNIST était donc le fait d'avoir des images en couleurs RGB. La taille moindre du réseau et des données à traiter permettait de gagner du temps et de conduire plus facilement des expérimentations variées. Ainsi, nous avons expérimenté ici divers réseaux convolutifs, à savoir certaines versions « maison » de taille modeste ainsi que des variantes autour de VGG16 : chargé pré-entraîné ou non, en figeant comme sur « 102 flowers » la partie convolutive (utilisée pour la *feature extraction*), ou en

64. Voir <https://www.cs.toronto.edu/~kriz/cifar.html>

entraînant le modèle complet (*fine tuning global*)<sup>65</sup>. Si la démarche a été formatrice, les résultats n'ont pas été pour autant transférables tels quels. En effet, les formats d'images, le nombre de catégories étaient très différents. C'est pourquoi nous ne les détaillons pas.

Pour pouvoir traiter de nombreuses combinaisons de paramètres tout en récoltant les résultats dans des fichiers CSV, exploitables pour diverses observations et statistiques, nous avons construit un script Python dédié. Les différentes valeurs des hyper-paramètres y sont indiquées dans des listes qui sont ensuite itérées.

```
# Hyperparameters to test
lr_range = [1e-4, 5e-5]           ##### <== choice (enumeration)
batch_size_range = [64, 32, 8, 4, 1] ##### <== choice (enumeration)
epochs = 10                         ##### <== choice (1 value=max)
# Number of iterations for each parameter
iter = 3                            ##### <== choice (single value)
```

Les programmes mis en œuvre sont consultables, accompagnés d'une courte explication de leur utilisation, sur <https://github.com/jmg-74/exam>. Le contexte étant établi, voyons maintenant à quoi nous sommes arrivé.

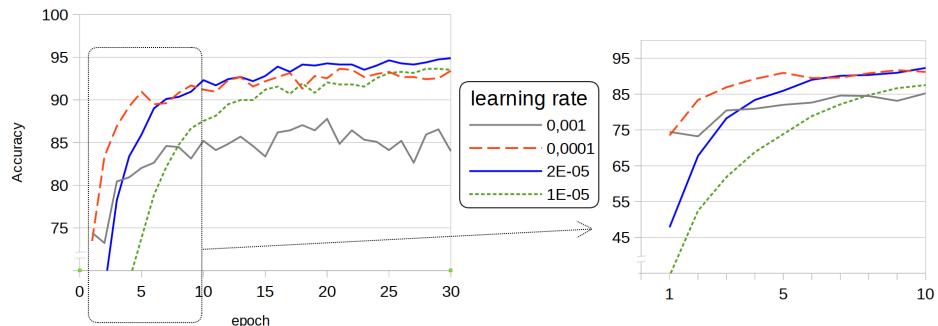
#### 4.2.3 Résultats obtenus

Pour commencer, l'exemple de MNIST nous a fourni une référence. Avec les réglages par défaut, à savoir une taille de *mini-batch* de 64 et un taux d'apprentissage de 0,1 la précision obtenue sur le modèle de base, au bout de 10 *epochs* est en moyenne de 99,0 %. Au passage, c'est assez incroyable, vu la simplicité du réseau neuronal employé. Elle tombe à 91,2 % en moyenne, soit une baisse de 8 points (8,1 %), quand le modèle est converti en version DP. Et les résultats sont alors bien plus irréguliers, puisque l'écart-type est plus que triplé.

En ce qui concerne notre modèle de départ (sans DP) de « 102 flowers », l'*accuracy* augmente rapidement, grâce au modèle pré-entraîné : les 80 % sont atteints après une seule *epoch*, pour certains paramétrages. En général après une vingtaine de cycles, on peut dépasser régulièrement 94 % (sans atteindre 95 % sauf cas exceptionnel).

C'est en faisant varier les hyper-paramètres que nous déterminons les valeurs les plus adaptées. Nous retrouvons ainsi leur impact sur la phase d'apprentissage :

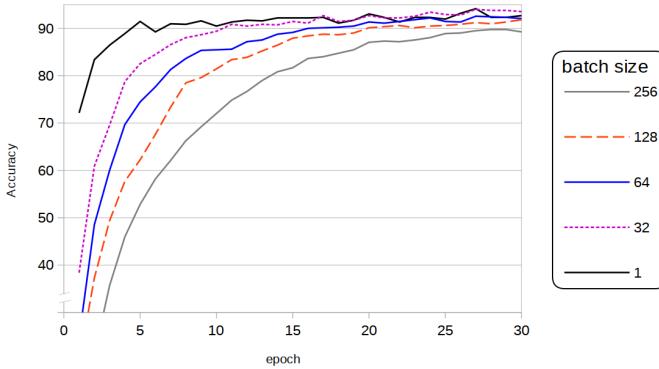
- Un ***learning rate*** plus élevé accélère le démarrage, mais devient nuisible à la convergence (à cause des « oscillations » autour du minimum causées par le pas trop important). Voici par exemple une représentation de l'*accuracy* constatée en fonction du nombre d'*epochs*, pour un paramétrage donné :



65. Les termes anglophones cités le sont car ce sont eux qu'on rencontre généralement.  
<https://developers.google.com/machine-learning/glossary> constitue une source intéressante en cas de doute.

La valeur  $2 \cdot 10^{-5}$  paraît ici optimale si l'on n'est pas limité en nombre de cycles.

- Une **taille de *mini-batch*** plus faible rend également l'apprentissage plus rapide, mais il semble alors moins régulier. On pouvait s'y attendre, puisque les mises à jour des paramètres internes du modèle sont effectuées sur une quantité réduite de données. L'aléa dû au petit nombre d'entrées prises en compte à chaque mise à jour agirait comme une sorte de bruit, limitant l'*over-training*. Ainsi, il est réputé favoriser la généralisation. Il convient toutefois de rester prudent avant de tirer des conclusions hâtives. D'ailleurs, les explications divergent et les spécificités de chaque contexte peuvent être importantes. Une approche empirique, au cas par cas, garde tout son sens quand tant de facteurs interviennent.



Contrairement à MNIST, le **passage en version DP** s'avère beaucoup **plus préjudiciable à la précision**. Nous avons réalisé énormément de tentatives, là encore en partie en balayant grâce à un script plusieurs listes de configurations envisageables. Il s'est avéré impossible d'atteindre une précision dépassant nettement 55 %, sans que la valeur  $\epsilon$  mesurant la perte de confidentialité soit trop grande pour offrir encore une réelle garantie. Vu les premiers résultats, nous nous sommes borné à travailler avec  $\delta = 10^{-4}$ , qui ne serait pas suffisamment petit pour des données ultra-sensibles (mais qui n'est pas vide de sens, puisque cela reste inférieur à l'inverse de la taille du *dataset* d'entraînement). Avec  $\delta = 10^{-5}$  par exemple,  $\epsilon$  augmenterait de l'ordre de 10 % pour ses grandes valeurs (constatées dans notre contexte) à 12 % pour les faibles, comme on peut le vérifier sans peine (*cf.* 4.1.1).

Nous sommes loin des résultats obtenus sur MNIST. Cependant, il faut relativiser et bien mesurer la différence entre les deux : nous l'avons vu, les données traitées sont plus complexes et surtout, il y a ici 10 fois plus de catégories. Une précision de 55 % est certes rédhibitoire pour certaines utilisations. Mais elle est 56 fois plus précise qu'une réponse au hasard parmi les 102 catégories, là où le taux obtenu en DP pour MNIST est environ 9 fois plus élevé que celui d'un tirage aléatoire. Le défi était donc d'une toute autre nature !

On peut parfois présupposer de l'effet de chacune des valeurs modifiables. Mais ici, c'est difficile comme nous l'avons remarqué précédemment, surtout pour un novice. Nous avons donc principalement procédé de manière empirique, en faisant varier un paramètre à la fois et en observant l'effet obtenu concernant l'*accuracy*.

- Augmenter la **taille de *batch*** semblait bénéfique à la précision. Dans le même temps, il fallait augmenter le **bruit ajouté** pour limiter  $\epsilon$ . Cela nous ramène à la représentation d' $\epsilon$  en fonction de ces deux paramètres dans notre carnet Jupyter, qui permet de visualiser l'importance relative de chacun de ces hyper-paramètres. Baisser en parallèle le **taux d'apprentissage** semblait également préférable.

<i>Batch Size</i>	<i>Noise Multiplier</i>	<i>Seuil de Clipping</i>	<i>Learning Rate</i>	<i>Nombre d'epochs</i>	<i>Coût DP (<math>\epsilon</math>)</i>	<i>Accuracy</i>
64	1	10	0,0008	150	8,3	57 %
64	1	10	0,0008	143	8,1	56,8 %
64	1	10	0,0008	125	7,5	56,7 %
64	1	12	0,0007	120	6,2	56,1 %
64	1	10	0,0007	105	5,8	54,4 %
32	0,9	8	0,0007	97	5,4	50,6 %
64	1,2	14	0,0007	105	5	50,4 %
64	1,2	14	0,0007	102	4,9	50 %
32	0,9	8	0,0008	95	5,3	49,8 %
32	0,9	8	0,0007	77	4,8	49 %
64	1,2	14	0,0007	95	4,7	48,3 %
32	0,9	8	0,0008	60	4,2	46,1 %
64	1,2	14	0,0007	69	4	43,6 %

TABLE 1 – Résultats sur « 102 flowers » en version DP.

- Le rôle de la **taille de la couche cachée** d’entrée du perceptron n’était pas évident (paramètre `--hidden-units`). En dessous de 256, la précision chutait. Mais au dessus de 512 qui est apparue au final la valeur adaptée à la plupart des combinaisons, il ne semblait pas y avoir de gain notable.  
Sans compter qu’augmenter cette valeur allonge considérablement le temps de calcul. Et dans le même temps, cela a participé aux dépassements de la mémoire du GPU déjà évoqués. C’est pourquoi nous avons finalement figé à 512 *hidden units* cette couche de notre réseau neuronal.
- Nous avons également tenté d'**ajouter une couche cachée** au perceptron, pour augmenter l’expressivité de notre modèle. Mais cela n’a pas eu d’effet clairement observable et nous sommes donc revenu à la version initiale.
- Enfin (au départ dans le cadre de nos réflexions sur la consommation de mémoire), nous avons **réduit la largeur et la hauteur des images** d’un facteur 2, voire 4. Le second s’est avéré excessif, mais le 2 a permis de gagner en précision et c’est cette valeur que nous avons conservé au final.
- Une autre piste a été effleurée : **utiliser un autre optimizer**, comme Adam. Mais d’une part nous n’avons pas obtenu de nets progrès et les valeurs des différents hyper-paramètres qui semblaient adaptées en pratique étaient différentes. C’était notamment le cas du taux d’apprentissage. Ce constat nous empêchait de bénéficier de l’expérience acquise. D’autre part, cela nous emmenait trop loin, car il aurait fallu étudier plus en détail la transformation en version DP de ces nouveaux *optimizers*, en vérifiant notamment si les calculs de budget associés restaient valables ou non. Nous avons donc jugé plus raisonnable de garder de côté comme future ouverture cette voie d’exploration et sommes donc revenus au simple SGD.

Quelques résultats sont donnés dans le tableau 1, pour 512 unités sur la couche cachée du perceptron.

Partant du constat qu’augmenter la taille de *mini-batch* semblait bénéfique à la précision obtenue, nous avons continué dans ce sens. C’est ainsi que nous sommes

arrivé lors de certaines tentatives, comme mentionné précédemment, à dépasser la capacité mémoire du GPU portant doté de 12 Go. Cela nous a interpellés et nous avons décidé de mieux comprendre pourquoi les besoins étaient si importants et sur quelles variables il était possible de jouer pour éviter ces blocages.

### 4.3 Analyse des besoins en mémoire sur le GPU

#### 4.3.1 Méthode générale

Nous avons procédé par étapes, en auditant le code.

Le nombre d'octets occupés à un moment donné sur la carte désignée par la variable `device` est renvoyé par `memory_allocated(device)` (fonction du module `torch.cuda` à importer). Le fait que le *garbage collector* libère la mémoire qui n'est plus utilisée quand bon lui semble n'a pas entravé nos mesures. En effet, au départ on procède essentiellement en partant d'un support vide qui est chargé par étapes. Ce n'est qu'après un cycle « phase directe + *backpropagation* » que des emplacements utilisés par les premières données sont libérés pour charger les informations suivantes. Comme la taille de *mini-batch* est constante, on revient à chaque fois au même niveau qu'à la fin du traitement du premier lot. La mesure peut donc se limiter à cette première itération, le seuil observé n'ayant pas de raison d'être dépassé par la suite.

- Pour commencer, nous avons procédé par tâtonnement en suivant le volume de mémoire consommée sur GPU à différents stades de l'apprentissage du modèle. Cela nous a permis de **déterminer les différentes étapes** où un chargement supplémentaire est effectué.
- Vu le nombre imposant de combinaisons d'hyper-paramètres possibles, une observation manuelle n'étant pas réalisable. Nous avons donc écrit un script Python similaire à celui que nous avions utilisé pour suivre les précisions obtenues. Mais même l'appel à `torch.cuda.empty_cache()` semblait vain pour revenir à une situation vierge entre deux modèles successifs et il nous a semblé indispensable de relancer le *kernel* Python<sup>66</sup> entre chaque essai.

Nous avons alors créé un **programme Python qui se borne à scruter un cycle unique d'apprentissage** du premier *mini-batch*, pour une configuration donnée, récupérable via les arguments de la ligne de commande. Il ajoute ses résultats, constitués des relevés de mémoire aux différents stades-clés, dans un fichier CSV. Et c'est un **script Bash** qui assure le balayage des différentes possibilités envisagées et lance pour chacune ce programme Python.

```
#!/bin/bash
for HU in 32 64 512 ; do
    for BS in $(seq 2 1 7) ; do
        for F in 1 2 4 ; do
            ./flowers_mem_monitor.py --hidden-units $HU \
                --batch-size $BS -f $F
        done
    done
done
```

---

66. Voir par exemple la discussion sur le forum suivant : <https://forums.fast.ai/t/clearing-gpu-memory-pytorch/14637/2>

- La phase suivante a consisté à exploiter ces données à l'aide d'un tableur (en l'occurrence LibreOffice) <sup>67</sup>.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O					
1	GPU memory during training				(Bytes)	/ Tot.		Err. (%)	(kB)	Err. (%)	(kB)	(Bytes)	/ Tot.							
2					1. Model load	1,26 %		-1,21 %		Estimated to interp.	to 1	2. Images / tags load Delta	0,01 %	BS	I					
3					26	64	2	N	62 397 952	78,07 %	62 536 185	0,22 %	63 386 062	1,58 %	965					
4	HU	BS	F	DP	5	26	64	2	62 397 952	78,07 %	62 536 185	0,22 %	63 386 062	1,58 %	965	72 032 256	9 634 304	12,05 %	18	J
5	26	64	2	N	6	26	64	2	62 397 952	74,59 %	62 536 185	0,22 %	63 386 062	1,58 %	965	72 032 256	9 634 304	3,79 %	18	J
7	28	64	2	N	8	28	64	2	62 599 680	77,54 %	62 738 333	0,22 %	63 588 836	1,58 %	966	72 233 984	9 634 304	11,93 %	18	J
9	30	64	2	N	10	28	64	2	62 599 680	23,38 %	62 738 333	0,22 %	63 588 836	1,58 %	966	72 233 984	9 634 304	3,60 %	18	J
11	32	1	2	N	12	30	64	2	62 800 896	77,92 %	62 940 481	0,22 %	63 791 610	1,58 %	967	72 435 200	9 634 304	11,82 %	18	J
13	32	1	2	N	14	32	64	2	62 800 896	22,31 %	62 940 481	0,22 %	63 791 610	1,58 %	967	72 435 200	9 634 304	3,42 %	18	J
15	4096	128	2	N	16	6162	4096	128	472 932 864	27,32 %	473 906 551	0,21 %	476 031 152	0,66 %	3 028	492 201 472	19 268 608	1,11 %	32	Z
17	4096	256	2	N	18	6163	4096	256	472 932 864	27,02 %	473 906 551	0,21 %	476 031 152	0,66 %	3 028	511 470 680	38 537 216	2,20 %	64	Z
19	4096	512	2	N	20	6164	4096	512	472 932 864	26,43 %	473 906 551	0,21 %	476 031 152	0,66 %	3 028	550 531 584	77 598 720	4,34 %	128	Z
21	4100	64	2	N	22	6165	4100	64	472 933 376	27,46 %	474 310 847	0,29 %	476 436 700	0,74 %	3 211	482 567 680	9 634 304	0,56 %	18	J

Des représentations graphiques de la mémoire consommée en fonction de chacune des variables susceptibles de l'influencer donnaient une première idée de la nature de cette dépendance. Mais des irrégularités ponctuelles ont rendu inefficace une pure régression numérique, d'autant que ça n'est pas une tendance mais plutôt un plafonnement que nous recherchions. Nous avons donc procédé en grande partie par essais successifs, en affinant progressivement la formule obtenue, de façon à ce qu'elle majore le besoin en mémoire sans trop s'éloigner des observations.

Nous avons consigné des mesures aussi bien sur la forme « classique » que sur la variante incluant la garantie de confidentialité différentielle. Cela a permis de se rendre compte du surcoût occasionné par le passage d'une forme à l'autre.

#### 4.3.2 Quelques précisions utiles

Avant de passer aux résultats proprement dits, ajoutons quelques remarques.

Certaines **valeurs** peuvent paraître **un peu extrêmes ou particulières** et dès lors moins utiles. Nous avons en effet procédé à ce *monitoring* avant d'avoir obtenu des résultats un tant soit peu probants au niveau *accuracy*, sans avoir encore d'idée de ce que pourrait être un ordre de grandeur raisonnable pour chacun des hyper-paramètres. Les conserver a tout de même du sens : les tendances étaient plus faciles à observer sur un domaine large. Et le fait de valider les formules sur un nombre important de combinaisons renforce la confiance que l'on peut avoir dans ces estimations.

Par ailleurs, il semble **conseillé de s'en tenir aux puissances de 2** pour la taille des *mini-batches*<sup>68</sup>, pour faciliter les optimisations des bibliothèques de calcul matriciel utilisées en arrière-plan (nous avons vu en effet dans la partie théorique que le fonctionnement d'un neurone pouvait se ramener à ce type d'opérations). Nous n'avons pas pris en compte cet élément, qui nous avait échappé au moment des premiers relevés.

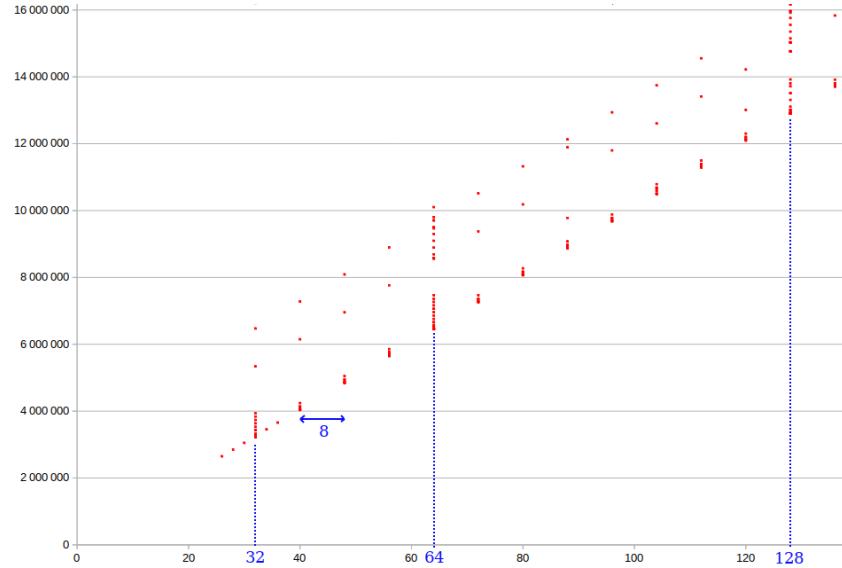
Cela a eu l'intérêt de nous montrer la présence d'**étranges irrégularités** au niveau de la consommation de mémoire, par endroits (en tout cas pour les valeurs *observées*, car il reste toujours envisageable, bien qu'improbable, qu'il s'agisse d'un artefact dû à notre méthode de relevé). Au niveau de certains seuils, qui justement coïncident avec les puissances de deux, le besoin augmente brutalement, semblant

67. Nous avons enregistré et exploité plus de 6 160 enregistrements, le fichier tableur au format Open Document (LibreOffice Calc) est accessible depuis <https://github.com/jmg-74/exam/docs>

68. Voir entre autres <http://www.deeplearningbook.org/contents/optimization.html> qui le rappelle, même si l'y a pas forcément unanimité, cf. <https://www.quora.com/Should-I-use-powers-of-2-when-choosing-the-size-of-a-batch-size-when-training-my-Neural-Network>

doubler, pour diminuer ensuite. Cela passait inaperçu sur nos premières mesures, exclusivement basées sur des valeurs en  $2^n$  !

Sur la représentation suivante, les points rouges indiquent des quantités de mémoire occupée pour l'une des étapes détaillées dans la suite, ici en fonction du nombre de neurones (*hidden units*) sur la couche du perceptron, chacun pour une combinaison différente des autres paramètres. Nous avons ajouté en bleu des indications montrant l'importance des puissances de 2 en lien avec les « bonds » observés.



Cela évoque fortement l'implémentation du type `list` de Python : un tableau de taille fixe qui est alloué, ce qui permet de bénéficier des avantages de cette représentation, comme la lecture et la modification de tout élément en temps constant. Et quand il est plein, pour dépasser la rigidité d'une telle représentation, un espace deux fois plus grand est utilisé. Les valeurs d'origine y sont alors copiées, ce qui crée ponctuellement un surplus d'utilisation du processeur. Cet effet de palier est préjudiciable, mais son coût est largement compensé en moyenne, précisément du fait des seuils limités aux tailles en  $2^n$ . On conserve ainsi une complexité moyenne d'accès qui reste proche de celle des tableaux de taille fixe, tout en gagnant la souplesse des listes chaînées de taille variable. Vu la filiation, il y a fort à parier que PyTorch utilise la même stratégie pour ses objets `Tensor`.

En tout cas, d'un point de vue pratique, **quand on constate un dépassement de la mémoire disponible**, cela donne une **stratégie pour surmonter cette limitation** : il peut être intéressant de tenter des valeurs (de *batch size* notamment) légèrement inférieures aux puissances de 2, quitte à perdre éventuellement un peu au niveau des optimisations calculatoires.

Revenons à présent à une vue plus globale, indépendamment de ces irrégularités.

#### 4.3.3 Modélisation

Nous avons déterminé quatre phases importantes. Les endroits où l'on relève l'occupation de la mémoire sur le GPU sont repérables dans le code source de `flowers_mem_monitor.py`, au niveau des appels à la fonction `_mem_monitor()`. Cette mesure est effectuée juste après l'instruction qui mobilise davantage de RAM, ce qui permet d'établir facilement le lien entre les deux.

Les variables dont dépend la quantité de mémoire consommée seront notées respectivement *HU* pour *hidden units* (nombre de neurones sur la première couche du perceptron), *BS* pour la (*mini*)-batch size, *F* pour le facteur de réduction de chacune

des dimensions des images (1, 2 ou 4) et  $DP$  pour la prise en charge de la protection ou non (Y/N) de la confidentialité différentielle. Les calculs sont exprimés par défaut en octets.

- La **première étape** correspond au **chargement du modèle**, de ses paramètres internes, sur le GPU. En toute logique, on retrouve un accroissement proportionnel à  $HU$ , aux irrégularités près (dont nous venons de parler).

Ainsi,  $(101\ 387\ HU + 60\ 750\ 000)$  majore la mémoire consommée, en englobant les « sauts » précédemment mentionnés (le surcoût maximum par rapport à la valeur constatée est de 3,4 Mo, soit moins de 1,6 %).

- La **deuxième augmentation** des besoins de mémoire vive intervient au **chargement des images** (et étiquettes). On retrouve un surplus par rapport à l'étape précédente qui est généralement proportionnel à  $BS/F^2$ , ce qui semble logique. Mais le quasi-doublement local (d'une partie ou de la totalité du besoin supplémentaire) aux seuils évoqués précédemment rend plus difficile d'établir une formule unique, qui surestime d'un facteur proche de 2 un grand nombre de situations.

La demande s'élève ainsi à  $(588 \times 1\ 024\ k \times BS^2/F)$  où  $k \in [1; 2[$ . Empiriquement, il est possible d'affiner la valeur obtenue, sans trop compliquer, en utilisant

- $(1\ 176\ BS^2/F)$  si  $BS^2/F \in ]1,625 ; 2,5[ \cup ]0 ; 0,1[$ ,
- $(820\ BS^2/F)$  si  $BS^2/F \in [2,5 ; 3,5[ \cup \{16\}$ ,
- $(615\ BS^2/F)$  si  $BS^2/F = 20$  ou  $30$ ,
- $(600\ BS^2/F)$  pour toutes les autres valeurs, les plus nombreuses.

On peut remarquer que  $588 \times 1\ 024 = (224^2 \times 3) \times 4$ , ce qui montre le lien avec le volume de l'image (le poids de l'étiquette est négligeable).

- L'analyse de la **troisième phase** demande de séparer les cas, selon la prise en charge ou non de la DP. Elle correspond à la **retro-propagation du gradient** de la fonction de coût (*cf. loss.backward()* dans le code).

L'élévation de la mémoire nécessaire vaut moins de  $(101\ 000\ HU + 1\ 339\ 400)$  en version « non DP ».

Avec la DP, on peut utiliser comme valeur de base  $(101\ 000\ HU + 7\ 489\ 400)$  à laquelle on ajoutera une pénalité dans certains cas particuliers, où  $HU$  est multiple de 8 et inférieur à 512 :

- si  $HU < 128$ ,  $88\ 000\ BS$ ,
- sinon si  $HU < 240$ ,  $18\ 850\ 000$ ,
- sinon  $5\ 821\ 000$ .

Il s'agit ici d'une estimation empirique qui évite l'ajout d'une constante élevée dans la formule, pour couvrir toutes les situations. Mais utiliser une expression unique ici aurait été possible également.

Le **surcoût de la DP** est donc au minimum de 6,15 Mo mais ne dépasse pas 25 Mo, sur cette partie. Cela reste donc négligeable par rapport aux besoins globaux qui posent problème (de l'ordre de 12 Go voire 16 Go en ce qui nous concerne).

- Pour terminer, c'est la **mise à jour des paramètres synaptiques** qui augmente encore les besoins (*cf. optimizer.step()*).

On consomme moins de  $(224\,105\,HU)$  octets en version « standard » (surestimé jusqu'à 11,2 % par ce calcul) et  $[108\,500(2 + BS)\,HU]$  en version DP.

On peut donc estimer à  $(108\,500\,BS \times HU)$  ce second surcoût associé à la confidentialité différentielle. Pour les configurations que nous avons largement utilisées, où  $HU = 512$ , cela représente  $55,552\,BS$  Mo. Soit 889 Mo pour des *mini-batches* de taille 16, ce qui cette fois-ci est loin d'être négligeable.

De telles formules permettent des estimations pour des cas non envisagés pour notre modèle, mais ne sont pas aisément généralisables. Les établir demande un temps conséquent qui est loin de compenser celui perdu en essais avortés, puisque le dépassement éventuel qui conduit à un plantage intervient au début de l'entraînement. Évidemment, retrouver de telles estimations sur un autre réseau neuronal serait bien plus rapide, maintenant qu'on a l'idée du type de fonction à utiliser pour chaque phase, ainsi que le phénomène de « sauts » ponctuels à prendre en compte. Mais **ces résultats ne sont pas réellement utiles en tant que tels, mis à part l'évaluation du surplus lié à la DP.**

Par contre, cette tentative de modélisation a eu l'intérêt d'aider **à mieux cerner les besoins pour chacune des étapes clés** de l'entraînement. Et de **mettre au jour les brusques doublements locaux** des besoins supplémentaires pour certaines étapes. Cela devrait permettre de repousser l'obstacle en cas de dépassement de capacité, en s'éloignant des valeurs qui en sont la cause.

Nous avons également **établi que la DP entraînait un surcoût conséquent** en termes de mémoire, qui augmente à peu près linéairement avec la taille des *mini-batches*.

Ajoutons que nous avons observé que **les parts relatives de la mémoire consommées par chacune des phases varient énormément** d'une situation à une autre. Il semble donc vain de vouloir établir une tendance générale à ce sujet. Nous renvoyons le lecteur intéressé au fichier tableau rassemblant les données ou directement aux calculs synthétisés ci-dessus pour le constater.

**Un dernier point nous a interpellé.** En effet, nos calculs montrent des besoins maximaux de l'ordre de 6 Go. Or, les paramétrages entraînant un dépassement de la RAM de 12 Go du GPU employé n'étaient pas beaucoup plus exigeants en termes de mémoire que ces derniers. Il semble qu'il y a là aussi un phénomène de doublement soudain, à un moment donné, du besoin d'espace sur le GPU. C'est une **piste de travail intéressante**, car comprendre ce phénomène **pourrait permettre d'entraîner des modèles plus exigeants, à matériel égal**.

PyTorch parallélise-t-il automatiquement certaines opérations quand elles opèrent de manière indépendante ? Est-ce dû à un manque de réactivité du ramasse-miettes ? Cela restera un problème ouvert, à notre niveau, mais à lui seul il donne davantage de sens encore à nos observations dans ce domaine.

## 5 Conclusion

Nous voici au terme de cette présentation. Son élaboration nous a conduit à explorer des domaines particulièrement motivants.

La confidentialité différentielle est devenue incontournable quand il s'agit de respecter la sphère privée. Elle a trait à des thématiques qui nous tiennent à cœur à titre personnel et nous concernent tout particulièrement dans notre métier d'enseignant.

La pratique du *deep learning* et les recherches dans ce domaine nous ont apporté des éléments pour mieux nous représenter ce concept, dont on parle tant mais souvent si mal, faute de bien le comprendre. Elles nous ont donné les bases pour aller de l'avant dans les domaines connexes.

Les ponts établis entre les deux thématiques ont montré qu'il existe des voies pour utiliser ces technologies innovantes dans le respect des citoyens, pour le bien du plus grand nombre. Mais également que leur mise en œuvre est loin d'être automatique, qu'elle prendra du temps et se heurtera à des obstacles. Au delà de la mise en avant commerciale de « concepts magiques » comme la confidentialité différentielle, il reste indispensable d'approfondir, de vérifier que leur application est rigoureuse. La compréhension acquise nous y aidera.

D'autre part, nous espérons que notre exposé aura pu apporter un éclairage sur certains sujets. En particulier, nous serions heureux et fier qu'il puisse être utile à l'équipe qui nous a encadré et nous a offert ce cadre de travail. Ce serait un juste retour, si modeste soit-il. C'est également dans cet état d'esprit que nous avons rédigé notre *Jupyter notebook*.

De notre côté, cette expérience aura apporté également globalement un bon nombre d'éléments intéressants, au delà des notions acquises. Les études d'articles, les discussions avec Jean-François Couchot, les échanges épistolaire avec d'autres chercheurs nous ont permis d'avoir enfin une image de ce que pouvait être l'univers de la recherche, de l'ambiance qui peut y régner. L'utilisation plus intensive que d'ordinaire d'outils comme Git ou L<sup>A</sup>T<sub>E</sub>X pour ne citer qu'eux et la lecture d'un nombre conséquent d'articles et de documentations en langue anglaise nous ont fait progresser dans ces domaines.

L'intégration de petites productions de notre cru au sein de projets d'envergure comme PyTorch-DP a été, il faut bien l'avouer, un élément de satisfaction. Il en est de même pour les quelques détections ou corrections de *bugs* sur GitHub, ou une coquille relevée dans un article.

Pour terminer, rappelons que nous proposons en annexe le résultat de l'étude que nous avions conduite sur la méthode RAPPOR. Il s'agit d'une autre approche possible pour garantir la confidentialité différentielle dans un contexte différent, comme nous l'avons déjà expliqué. Nous avons choisi de ne pas l'intégrer directement au reste de l'exposé pour ne pas alourdir un contenu déjà dense. Et parce que nous avions axé nos investigations sur les outils de classification automatique.

## Références

- [ACG<sup>+</sup>16] Martin Abadi, Andy CHu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. <https://arxiv.org/abs/1607.00133v2>, 2016.
- [BDMN05] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy : The sulq framework. *PODS*, 2005.
- [BIZ15] Graeme Blair, Kosuke Imai, and Yang-Yang Zhou. Design and Analysis of the Randomized Response Technique. *Journal of the American Statistical Association* 110(511) :1304–1319, 2015.
- [CGR19] Jean-François Couchot, Christophe Guyeux, and Guillaume Royer. Anonymously forecasting the number and nature of firefighting operations. *23rd International Database Engineering & Applications Symposium*, 2019.
- [Dal77] Tore Dalenius. Towards a methodology for statistical disclosure control. *Statistik Tidskrift* 15, 429-444, 2-1, 1977.
- [DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. *Theory of Cryptography, Third Theory of Cryptography Conference.*, 2006.
- [DN03] Irit Dinur and Kobbi Nissim. Revealing Information while Preserving Privacy. *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, p. 202–210, 2003.
- [DR14] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science Vol. 9 : No. 3-4, pp 211-407*, 2014.
- [Dwo06] Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, Lecture Notes in Computer Science, pages 1–12. Springer Verlag, 2006.
- [EPK14] Lfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. RAPPOR : randomized aggregatable privacy-preserving ordinal response. *Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, pages 1054–1067*, 2014.
- [FLJ<sup>+</sup>14] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmaco-genetics : An end-to-end case study of personalized warfarin dosing. *Proceedings of the 23rd USENIX Conference on Security Symposium, SEC’14*, 2014.
- [HGH<sup>+</sup>14] Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C. Pierce, and Aaron Roth. Differential privacy : An economic method for choosing epsilon. *Proceedings of 27th IEEE Computer Security Foundations Symposium (CSF)*, 2014.
- [LBD<sup>+</sup>89] Yann LeCun, B Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *AT&T Bell Laboratories*, 1989.
- [LMMP59] J.Y. Lettvin, H.R. Maturana, W.S. McCulloch, and W.H. Pitts. What the frog’s eye tells the frog’s brain. *Proceedings of the IRE, Vol. 47, No. 11, pp. 1940-51*, 1959.
- [Mir17] Ilya Mironov. Renyi differential privacy. <https://arxiv.org/abs/1702.07476>, 2017.
- [MTZ19] Ilya Mironov, Kunal Talwar, and Li Zhang. Rényi differential privacy of the sampled gaussian mechanism. <https://arxiv.org/abs/1908.10530v1>, 2019.

- [NS06] Arvind Narayanan and Vitaly Shmatikov. Robust De-anonymization of Large Datasets (How To Break Anonymity of the Netflix Prize Dataset). <https://arxiv.org/pdf/cs/0610105.pdf>, 2006.
- [NSW<sup>+</sup>17] Kobbi Nissim, Thomas Steinke, Alexandra Wood, Mark Bun, Marco Gaboardi, and Salil O'Brien, David R. and Vadhan. Differential privacy : A primer for a non-technical audience. *Privacy Tools for Sharing Research Data project at Harvard University*, 2017.
- [PBBML19] Vincent Primault, Antoine Boutet, Sonia Ben Mokhtar, and Brunie Lionel. The long road to computational location privacy : A survey. *IEEE Communications Surveys and Tutorials*, 21(3) : 2772-2793, 2019.
- [RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*. 323 (6088) : 533–536. doi :10.1038/323533a0. ISSN 1476-4687, 1986.
- [SS98] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information : k-anonymity and its enforcement through generalization and suppression. *Harvard Data Privacy Lab*, 1998.
- [Swe02] Latanya Sweeney. k-anonymity : A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5) :557–570, 2002.
- [SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. <https://arxiv.org/abs/1409.1556>, 2015.
- [TZJ<sup>+</sup>16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. <https://arxiv.org/pdf/1609.02943.pdf>, 2016.
- [Vad17] Salil Vadhan. *The Complexity of Differential Privacy*, pages 347–450. Springer, Yehuda Lindell, ed., 2017.
- [vEH14] Tim van Erven and Peter Harremoës. Rényi divergence and kullback-leibler divergence. <https://arxiv.org/abs/1206.2459v2>, 2014.
- [War65] Stanley L. Warner. Randomized Response : A Survey Technique for Eliminating Evasive Answer Bias. *Journal of the American Statistical Association*, 60 (309), 63–69, 1965.

# ANNEXES

## 6 Méthode RAPPOR – collecte des données

### 6.1 Vue d'ensemble

RAPPOR, pour *Randomized Aggregatable Privacy-Preserving Ordinal Response* est défini comme une technologie « permettant l'étude de la forêt des données des utilisateurs, en empêchant d'observer les arbres individuellement » [EPK14].

Il s'appuie sur un concept robuste, ne nécessite pas de tiers de confiance et il est distribué sous une licence libre permissive (Apache 2.0). On peut l'utiliser pour collecter des statistiques de catégories diverses, fréquences, histogrammes ou autres côté client, avec de solides garanties de respect de la confidentialité y compris lors de requêtes répétées à l'identique auprès d'un même individu, ce qui est notable.

L'utilisation typique est celle d'un fournisseur de service « cloud » voulant analyser les usages de ses usagers, notamment pour améliorer son produit et renforcer la sécurité des utilisateurs, tout en respectant leurs données individuelles. RAPPOR est alors installé côté client. Il a d'ailleurs également été inclus dans le navigateur Chrome, avec des objectifs similaires.

### 6.2 Algorithme fondamental

RAPPOR s'attache à protéger la confidentialité à la fois pour les requêtes uniques — via les réponses randomisées — mais aussi pour celles qui sont répétées dans le temps — en mémorisant une réponse randomisée qui sera reprise à la place de la vraie valeur avant d'être elle-même bruitée, en cas de requêtes réitérées. Dans ce dernier cas, un adversaire exploitant de manière statistique les résultats de questions successives identiques, même sans limitation, pourra au mieux retrouver la réponse bruitée conservée. Ainsi, la protection ne se limite pas à l' $\varepsilon$ -indiscernabilité qui alloue un budget de confidentialité s'amenuisant au fur et à mesure des requêtes.

Et les différents paramètres restent indépendants, ce qui permet un réglage fin en fonction du contexte. En effet, l'algorithme peut être découpé en trois parties relativement autonomes précédant la phase de réponse.

*Algorithme* : la donnée  $v$  de départ, à protéger, est écrite comme une chaîne de bits, sans contrainte particulière. RAPPOR s'exécute sur la machine du client et renvoie une donnée construite à partir de  $v$  au serveur — on verra dans un second temps en quoi celle-ci reste utilisable pour calculer des statistiques pertinentes. Les paramètres sont les suivants :  $k \in \mathbb{N}^*$ ,  $h \in \mathbb{N}^*$  et  $f, p, q$  qui sont des probabilités.

1. **Le signal** : la valeur  $v$  est hachée par un **filtre de Bloom**<sup>69</sup>  $B$  de taille  $k$  utilisant  $h$  de fonctions de hachage. On obtient ainsi un vecteur de  $k$  bits notés  $B_i$ .

Cette étape sécurise  $v$ , en fournissant un déni plausible qui peut être important.

---

69. Ce concept de 1970 est bien décrit sur le [Wikipedia](#) francophone et illustré de manière pédagogique par [geeksforgeeks.org](#).

Il s'agit de construire un mot  $B$  de  $k$  bits associés à une valeur  $v$  (ou plus largement un ensemble  $E_v$  de valeurs), par hachage de cette valeur (ou de celles de  $E_v$ ) : une ou plusieurs fonctions de hachage à valeurs dans  $\llbracket 1; k \rrbracket$  vont indiquer quels bits activer dans  $B$ . Pour toute valeur  $v'$  au format de  $v$ , on obtient un mot  $B'$  en lui faisant subir les mêmes opérations de hashage. Si l'un des bits activés dans  $B$  ne l'est pas dans  $B'$ , on déduit avec certitude que  $v' \neq v$  (ou plus généralement que  $v' \notin E_v$ , ce qui permet d'exclure une telle appartenance en temps constant, avec un espace consommé constant). Sinon, rien n'est certain. Dans notre cadre, c'est l'aspect « clapet anti-retour » qui est utilisé pour protéger  $v$ , au prix de l'acceptation d'éventuels « faux positifs ».

Là où la confidentialité différentielle qui va être utilisée dans les étapes suivantes apporte des garanties pour le pire des cas, ce filtre ajoute sa protection (du même type que le  $k$ -anonymat) pour le cas moyen.

2. **La réponse randomisée permanente** : on calcule les bits  $B'_i$  de ce vecteur de taille  $k$  ainsi :

$$B'_i = \begin{cases} 1 & \text{avec la probabilité } \frac{1}{2}f \\ 0 & \text{avec la probabilité } \frac{1}{2}f \\ B_i & \text{avec la probabilité } 1 - f \end{cases}$$

Cette valeur  $B'$  est mémorisée, afin de protéger des attaques longitudinales.

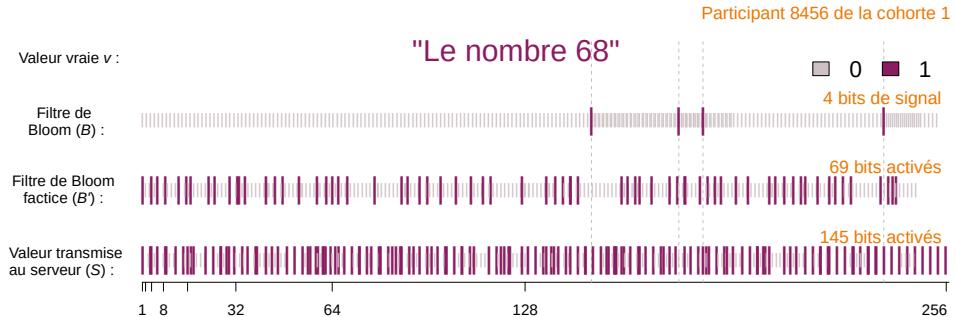
3. **La réponse randomisée instantanée** : on initialise un tableau  $S$  de  $k$  bits à 0. Puis chacun prend la valeur 1 avec la probabilité

$$\mathbb{P}(S_i = 1) = \begin{cases} q & \text{si } B'_i = 1 \\ p & \text{si } B'_i = 0 \end{cases}$$

C'est ici qu'est mise en œuvre l' $\varepsilon$ -indiscernabilité « classique ».

4. **Le renvoi au serveur** de la valeur  $S$ .

Nous nous permettons de reprendre telle quelle (*simplement traduite*) l'illustration faite par les auteurs de RAPPOR [EPK14], pour  $v = 68$ ,  $k = 256$ ,  $h = 4$  (et  $f = 0,5$ ,  $p = 0,5$ ,  $q = 0,75$ ) :



### 6.3 Versions modifiées

Cet algorithme reste adaptable selon le scénario :

- **Collecte unique** : par nature, la protection longitudinale est inutile. On peut sauter l'étape 2 (donc choisir  $f = 0$ ).
- **Collecte basique** : il s'agit du cas où l'ensemble de chaînes relevées est assez petit et où chacune peut être représentée par un bit unique dans un tableau (le genre d'une personne par exemple). Dans ce cas, l'utilisation du filtre de Bloom est inutile. On la remplacera par la traduction déterministe de chacune des valeurs en un unique bit de  $B$  (d'où  $h = 1$ ).
- **Collecte basique unique** : combinaison des précédentes, c'est le cas le plus simple envisageable.

## 6.4 Influence des paramètres

### 6.4.1 Epsilon

Notons que le choix de  $\varepsilon$  n'est pas trivial, faute d'être immédiatement interprétable : l'article de Justin Hsu *et al.* [HGH<sup>+</sup>14] en 2015 semble être le premier consacré à cet objectif et fait état de la complexité de la détermination de ce réel, en l'absence d'une sémantique évidente. Cet unique curseur doit en effet correspondre à des situations et des intérêts divers : l'analyste attend un maximum de précision de ses statistiques, tout en puisant le moins possible dans son « budget de confidentialité ». L'utilisateur qui cède ses données, même avec une contre-partie, désire la plus grande confidentialité possible. Et la prise de risque acceptable sera fonction de la nature des données à protéger.

Hsu constate ([HGH<sup>+</sup>14] tableau 1 page 24) que dans la littérature  $\varepsilon$  peut facilement varier de 0,01 à 10, sans qu'une justification ne soit toujours apportée ! Dans l'article d'Erlingsson qui présente RAPPOR [EPK14], c'est  $\varepsilon = \ln 3$  qui est choisi pour les expérimentations, à savoir la valeur associée au sondage randomisé historique de L. Warner, associé à une cote 3:1 à propos du pari sur une réponse « Oui » ou « Non ».

### 6.4.2 Utilisation de cohortes

Le but d'une étude conduite avec RAPPOR est généralement de déterminer quelles chaînes sont présentes et à quelle fréquence, dans la population étudiée. On cherche donc à réduire le taux de faux positifs (dus au filtre de Bloom). À cet effet, on partitionne aléatoirement la population sondée en  $m$  cohortes d'effectifs similaires. Chacune utilise un ensemble de fonctions de hachage qui lui est propre pour son filtre, afin de réduire les collisions de chaînes d'une cohorte à l'autre.

Les faux positifs augmentent quand  $m$  est trop petit, mais s'il est trop grand, chaque cohorte procure un signal insignifiant vu son faible effectif. Le choix de  $m$  est donc important.

### 6.4.3 Paramètres probabilistes et de configuration du filtre de Bloom

Les paramètres  $f$ ,  $p$  et  $q$  seront établis selon la valeur de  $\varepsilon$  désirée. On prendra en compte le fait que le taux de faux positifs (FDR, *False Discovery Rate*) augmente proportionnellement à  $f$ . En l'absence de données longitudinales, on a vu qu'on peut opter pour la version « collecte unique » de RAPPOR ( $f = 0$ ).

La taille  $k$  du filtre de Bloom, la quantité  $h$  de ses fonctions de hachages et le nombre  $m$  de cohortes doivent également être fixées *a priori*. Puisque ni  $k$  ni  $m$  n'interviennent au niveau de la détermination du cas le plus défavorable en termes de confidentialité, il seront choisis pour déterminer l'efficacité de la reconstruction du signal à partir de la sortie bruitée  $S$ . L'expérience indique que  $h$  semble être le seul réellement déterminant en la matière (la plus petite valeur testée, 2, semblant la meilleure au niveau utilisabilité compte tenu des autres paramètres choisis pour le test par Erlingsson[EPK14]).

Au final, aucune règle évidente et automatique ne s'impose. La pratique restera prépondérante pour affiner ces paramètres.

## 6.5 Confidentialité différentielle de RAPPOR

Revenons sur la solidité de RAPPOR face aux diverses menaces envisageables.

### 6.5.1 Garanties sur la réponse randomisée permanente

Intuitivement, les étapes 1. et 2. contrent efficacement les adversaires exploitant les données longitudinales. Car comme on l'a mentionné, une analyse statistique permettrait au mieux de faire fi de l'étape 3 et de remonter à  $B'$ , mais pas à  $v$ , ni même à  $B$ .

Plus précisément, on montre que ces deux manipulations satisfont l' $\varepsilon$ -indiscernabilité, pour  $\varepsilon = 2h \ln\left(\frac{1-\frac{1}{2}f}{\frac{1}{2}f}\right)$ , valeur qu'on notera  $\varepsilon_\infty$ . Elle ne dépend que de  $h$  et  $f$ . Une valeur faible de  $k$  renforce la sécurité en augmentant les collisions dans le filtre de Bloom — et risque donc de nuire à l'utilisabilité de  $S$  — mais ça n'est ni nécessaire ni suffisant pour garantir l' $\varepsilon$ -indiscernabilité).

Nous ne détaillons pas la preuve ; elle repose sur les probabilités conditionnelles et l'indépendance d'événements, les calculs déjà évoqués autour de la réponse randomisée, et le fait que pour des positifs, on a  $\frac{a+b}{c+d} \leq \max\left(\frac{a}{c}, \frac{b}{d}\right)$  ([EPK14] 3.1 page 5).

### 6.5.2 Garanties sur la réponse randomisée instantanée

Là encore, les deux étapes de randomisation forment une solide protection.

On remarque que la probabilité  $\mathbb{P}(S_i = 1|B_i = 1)$  qu'un bit soit à 1 sur la sortie  $S$  sachant qu'il l'est sur  $B$  vaut  $q^* = \frac{1}{2}f(p+q) + (1-f)q$  et que  $\mathbb{P}(S_i = 1|B_i = 1)$  vaut  $p^* = \frac{1}{2}f(p+q) + (1-f)p$ . Avec ces notations, il est alors prouvé que la réponse randomisée instantanée (étape 3) fournit la garantie de l' $\varepsilon_1$ -indiscernabilité, pour  $\varepsilon_1 = h \ln\left(\frac{q^*(1-p^*)}{p^*(1-q^*)}\right)$ . La démonstration est analogue à une partie de la précédente.

En cas de réponses multiples, déterminer  $\varepsilon_n$  doit s'effectuer en prenant en compte la connaissance déjà apportée à l'adversaire. Déterminer des stratégies pour majorer cette valeur reste un chantier ouvert, en tout cas en 2014.

Comme on pouvait s'y attendre au vu de l'algorithme, les garanties offertes sont solides. De plus, cela a pu être rigoureusement justifié. Reste à vérifier qu'il est possible de tirer suffisamment d'informations pertinentes de données qui ont subi un tel traitement et à indiquer comment s'y prendre.

## 7 Méthode RAPPOR – exploitation des résultats

On rappelle qu'en général, le but est de savoir quelles chaînes sont présentes et à quelle fréquence, dans la population analysée. Vu les différentes étapes de transformation, l'utilisation d'outils statistiques avancés s'avère nécessaire.

### 7.1 Étapes clés du décodage

- En notant, pour une cohorte  $j$ ,  $t_{ij}$  le nombre de bits d'indice  $i$  activés dans  $B$  et  $c_{ij}$  celui des bits  $i$  à 1 dans un ensemble de  $N_j$  réponses, on peut estimer le premier par

$$t_{ij} = \frac{c_{ij} - (p + \frac{1}{2}fq - \frac{1}{2}fp)N_j}{(1-f)(q-p)}$$

On note  $Y$  un vecteur de  $t_{ij}$ , pour  $i \in \llbracket 1; k \rrbracket$  et  $j \in \llbracket 1; m \rrbracket$ .

- On crée une matrice  $X$  de taille  $km \times M$ , où  $M$  désigne le nombre de chaînes candidates. Elle est peu dense, avec beaucoup de 0 et des 1 associés au hachage du filtre de Bloom de chacune des chaînes de chaque cohorte.

Une régression LASSO<sup>70</sup> permet d'ajuster un modèle  $Y \sim X$  et de sélectionner ainsi les chaînes candidates associées aux coefficients non nuls.

- L'estimation des décomptes, ainsi que l'erreur standard et la *p-value*, sont obtenues par la méthode des moindres carrés sur les variables sélectionnées.
- Par correction de Bonferroni<sup>71</sup> au seuil de  $0,05/M$  (*pour un choix classique de seuil  $\alpha = 0,05$* ), on ne garde que les valeurs significativement non nulles d'un point de vue statistique.  
On peut également contrôler le taux de faux-positifs parmi les résultats *a priori* significatifs (FDR) avec la procédure de Benjamini-Hochberg<sup>72</sup>

## 7.2 Évaluations expérimentales

La pertinence de RAPPOR a été montrée [EPK14] sur les cas suivants. Pour chacun, on a choisi  $p = 0,5$  et  $q = 0,75$  (pour obtenir  $\varepsilon = \ln 3$ ) :

- **Retrouver une distribution normale** : on a simulé une telle répartition aléatoire (avec  $\mu = 50$  et  $\sigma = 10$ ) sur 10 000 valeurs. Avec  $f = 0$ , on constate que 10 000 données rapportées sont insuffisantes. 100 000 réponses font clairement apparaître la courbe en cloche sur une représentation graphique et un million en donnent un tracé précis.
- **Avec une distribution exponentielle** : un million de rapports sont collectées, sur autant de chaînes aléatoirement échantillonées suivant une distribution exponentielle. On opte pour  $f = 0,5$ ,  $h = 2$ ,  $k = 128$  et  $m = 16$  cohortes. Après analyse avec la correction de Bonferroni, 47 chaînes sont estimées comme significativement présentes, dont 2 seulement sont des faux positifs. Les plus fréquentes sont particulièrement bien détectées ( $p\text{-value} < 10^{-10}$  pour le top 20,  $p\text{-value} < 10^{-31}$  pour le top 10) et toutes les chaînes présentes avec une fréquence supérieure à 1% sont détectées. Celles de la « queue » de la fonction de densité sont ignorées, protégées par le mécanisme de confidentialité.
- **Cas réel : noms des processus sous Windows.** Il s'agit de 186 792 relevés sur 10 133 machines, listant les processus les plus actifs, pour identifier les plus populaires et estimer la fréquence d'un exécutif malicieux particulier. Avec  $h = 2$ ,  $k = 128$ ,  $m = 8$ ,  $f = 0,5$  pour  $\varepsilon_1 \approx 1,07$ , 10 processus ont été identifiés (fréquences de 2,5% à 4,5%). La fréquence du *malware* a pu être estimée à 2,6%.
- **Second cas réel : pages d'accueil sous Chrome.** RAPPOR a été implémenté dans le navigateur, pour collecter quotidiennement auprès d'utilisateurs l'ayant accepté des réglages particuliers (qui peuvent être modifiés à l'insu de l'utilisateur par des logiciels malveillants, d'où l'intérêt d'un suivi), dont le choix de la page d'accueil. Avec  $k = 128$ ,  $h = 2$ ,  $m = 32$  cohortes,  $f = 0,75$

70. *Least Absolute Shrinkage and Selection Operator* est une méthode de régression qui fonctionne bien en grande dimension et permet de sélectionner au plus près possible un sous-ensemble restreint de variables par minimisation des carrés des résidus, ou tout autre méthode statistique analogue : [https://en.wikipedia.org/wiki/Lasso\\_\(statistics\)](https://en.wikipedia.org/wiki/Lasso_(statistics))

71. Le but de cette correction est d'éliminer des valeurs qui apparaissent comme potentiellement significatives, alors que cela n'est dû qu'à la multiplicité des tests sur les données traitées (un seuil de 0,05 n'est alors plus adapté).

Voir par exemple <https://www.stat.berkeley.edu/~mgoldman/Section0402.pdf>

72. [https://en.wikipedia.org/wiki/False\\_discovery\\_rate#Benjamini-Hochberg\\_procedure](https://en.wikipedia.org/wiki/False_discovery_rate#Benjamini-Hochberg_procedure) ou <http://www.biostathandbook.com/multiplecomparisons.html>

pour  $\varepsilon_1 \approx 0,53$ , sur une douzaine de millions de relevés, on obtient les résultats suivants : une page doit être signalée par environ 14 000 utilisateurs au moins pour être identifiée par RAPPOR ; 31 pages inattendues ont été relevées, certaines sur des domaines potentiellement malicieux. Seules 0,5% des URL candidates ont une présence suffisamment significative statistiquement, mais une fois réunies, elles représentent 85% des fréquences relevées.

On constate que RAPPOR est adapté à diverses études de tendance, de distribution, ou de mise en évidence de faits anormaux, à condition qu'ils soient suffisamment fréquents. En effet, par nature, les données peu probables seront noyées dans le bruit assurant la confidentialité différentielle.

### 7.3 Limites de RAPPOR

L'intérêt évident de cet outil et sa puissance prouvée n'en font pas une panacée. Il devra donc s'intégrer dans un ensemble de mesures diverses (celles déjà évoquées, ainsi que la décentralisation du stockage, sa limitation dans le temps, etc.)

En effet, certaines faiblesses subsistent. Par exemple, un utilisateur utilise parfois plusieurs terminaux. Cela permet en théorie de contourner la garantie de confidentialité par des requêtes sur différents canaux pour un même individu, même si en pratique il reste difficile de déterminer ces entrées associées. Un choix de cohortes trop nombreuses et donc petites facilite le repérage d'une personne. Tous ces points pourraient affaiblir la confidentialité, dans l'absolu. Plusieurs peuvent cependant être évités ou atténués par des mesures additionnelles.

Un point fort de RAPPOR est de pouvoir fonctionner côté client, en laissant potentiellement l'utilisateur d'un LBS paramétriser lui-même le risque admis. Mais cela reste très théorique car la sémantique de  $\varepsilon$  est hors de portée de la plupart des individus. Sans compter qu'au-delà des conséquences directes d'une perte de confidentialité, on ne réalise généralement pas tout ce qui peut en être déduit. L'utilisateur insuffisamment éclairé risque donc d'être contraint de s'en remettre au LBS et de lui faire confiance, alors que leurs intérêts ne convergent pas nécessairement.