Résumé de

Deep Learning with Differential Privacy

arXiv:1607.001333

1 Vue d'ensemble

Le *Machine Learning* (ML) à base de réseaux neuronaux repose sur l'exploitation de données d'apprentissage nombreuses, issues du monde réel et qui peuvent donc contenir des informations sensibles. Ces dernières ne devraient pas être récupérables à partir du modèle construit par le réseau de neurones. L'article date de fin octobre 2 016. Il analyse les techniques qui permettent de le garantir dans une certaine mesure et propose un outil pour comptabiliser le « budget de confidentialité » qu'on doit déterminer au départ et qui est consommé ensuite par l'utilisation des données d'entraînement.

La démarche proposée permet une utilisation plus parcimonieuse de ce budget, par rapoort aux approches décrites jusque là. La mémoire utilisée est moindre, grâce à une utilisation de *batches* de plus petite taille. La méthode reste adaptée aux fonctions d'objectif non convexes. La mise en œuvre s'appuie sur le *framework* `TensoFlow` développé par Google. Elle est illustrée sur les *datasets* publics classiques de catégorisation d'images que sont `MNIST` ()*reconnaissance de chiffres manuscrits sur des images monochromes* et `CIFAR-10` (*identification d'objets et d'animaux, sur des images en couleur*).

On peut noter que les applications de ML contiennent déjà fréquemment des éléments qui contribuent au respect de la confidentialité des données d'entraînement. Il s'agit notamment des traitements pour éviter le surentraînement (*overfitting*), sorte d'apprentissage par cœur stupide qui ferait focaliser le modèle sur des détails spécifiques aux exemples donnés pour l'apprentissage, en perdant de vue les caractéristiques réellement discriminantes dans le cas général. Mais même si le fonctionnement interne des réseaux neuronnaux est difficile à expliciter, certains ont montré qu'on pouvait parfois retrouver en partie des informations utilisées pour l'entraînement du modèle, comme des images sur un algorithme de reconnaissance faciale.

Pour rester dans un cadre réaliste, on considère ici un adversaire ayant accès aux réponses du modèle à ses requêtes et qui dispose potentiellement de données et d'une puissance de calcul sans limites. Il sera même censé

connaître complètement le mécanisme d'apprentissage, dont ses paramètres. Le cadre de la confidentialité différentielle est parfaitement adapté à cet effet.

1.1 Confidentialité différentielle

Les définitions essentielles sont rappelées : deux bases de données sont dites **adjacentes** quand elles ne diffèrent que d'un unique enregistrement, présent uniquement dans l'une des deux. On dit alors qu'un méchanisme randomisé $\mathcal{M} : \mathcal{D} \to \mathcal{R}$ garantit la **(ε,δ)-DP** (pour *Differential Privacy*, confidentialité différentielle), si quelles que soient les entrées d et d' adjacentes de $\mathcal{D}$ et les parties S et S' de $\mathcal{R}$, on a

$$\mathbb{P}\left(\mathcal{M}(d) \in S\right) \leqslant e^{\varepsilon}\, \mathbb{P}\left(\mathcal{M}(d) \in S'\right) + \delta$$

Il s'agit déjà d'un relâchement (C. Dwork 2 006), de la définition originale de la « pure » **ε-DP** (cas où $\delta = 0$) : la valeur de relâchement δ, typiquement inférieure à $\frac{1}{|d|}$, est la probabilité que la garantie de confidentialité différentielle soit mise en défaut. Le nombre ε peut être interprété comme un **budget de confidentialité**. [Ajout : *cette version relâchée a été introduite pour pouvoir s'adapter à l'ajout d'un bruit qui suit une loi de Gauss plutôt que de Laplace*].

La DP a des propriétés bien utiles dans le contexte du ML : **compatibilité avec la composition** (l'enchaînement de deux mécanismes, respectivement (ε,δ)-DP et (ε',δ)-DP, est $(\varepsilon+\varepsilon',\delta)$-DP), **confidentialité de groupe** (*dégradation progressive de la confidentialité quand l'effectif augmente*), la **robustesse vis à vis des informations auxiliaires** connues par l'adversaire, etc.

La **sensibilité** d'une fonction réelle déterministe sur $\mathcal{D}$, notée S_f, est le maximum pour des entrées d et d' adjacentes, de l'écart absolu en sortie $|f(d) - f(')|$. L'ajout à f d'un bruit aléatoire de loi normale centrée d'écart-type $S_f \cdot \sigma$, soit $\mathcal{N}(0,\, S_f^2 \cdot \sigma^2)$, garantit la (ε,δ)-DP si $\delta \leqslant \frac{4}{5}\exp(-(\sigma\varepsilon)^2/2)$ et $\varepsilon < 1$. Cet exemple montre qu'on dispose d'une infinité de paramétrages optimaux, représentables sur une courbe implicite.

1.2 Deep Learning

Les différentes couches des réseaux « profonds » du *Deap Learning* (DL) définissent des fonctions paramétrées associant les sorties aux entrées, comme composées de couches multiples. Ces blocs élémentaires sont des transformations affines, des fonctions non-linéaires simples (souvent ReLU *REctified Linear Unit* qui vaut 0 dans les négatifs et l'identité dans les positifs, ou sigmoïde $\sigma(x) = \frac{1}{1+\exp(-\lambda x)}$).

On définit une fonction d'objectif, ou de perte $\mathcal{L}$ (pour *loss*) qui pénalise l'écart aux résultats attendus, en sortie, pour des données d'entraînement.

Plus précisément, $\mathcal{L}(\theta)$ est la moyenne des pertes $\mathcal{L}(\theta,x)$ pour chacune des entrées x, sur l'ensemble des données d'entraînement considérées, pour des paramètres θ donnés. Entraîner le réseau neuronal revient à minimiser au mieux cette quantité, ce qui est difficile dans les réseaux complexes où $\mathcal{L}$ n'est généralement pas convexe.

En pratique on utilise souvent la descente de gradient stochastique (**SGD** ***Stochastic Gradient Descent***) : à chaque étape, on forme un lot B (on utilisera ici l'anglais *batch* pour bien identifier chaque entité sans confusion) d'entrées d'entraînement tirées aléatoirement, sur lesquelles on calcule le gradient moyen $\mathbf{g}_B = \frac{1}{|B|}\sum_{x\in B}\nabla\theta\mathcal{L}(\theta,x)$, qu'on utilise pour mettre à jour les paramètres de θ (d'un « pas » donné, suivant $-\mathbf{g}_B$). TensorFlow automatise les calculs de cette rétropropagation du gradient.

2 Approche choisie dans l'article

Ici, la SGD est adaptée pour garantir la DP (version DP-SGD). On explique comment gérer le budget de confidentialité (ε) et configuer les hyperparamètres (*on nomme ainsi ceux de l'algorithme, pour les distinguer des coefficients calculés par le réseau neuronal et adaptés au fur et à mesure de son apprentissage*).

2.1 Principe de la DP-SGD

Traiter en sortie le réseau vu comme « boite noire » à l'issue de son entraînement pour en garantir la confidentialité demande d'ajouter trop de bruit pour conserver un résultat utilisable, car la DP impose de considérer le pire cas. Une approche plus sophistiquée (*pas nouvelle, mais affinée notamment pour la comptabilité du budget de confidentialité*) est adoptée ici : intervenir à chaque phase du calcul du gradient.

Concrètement, dans l'**algorithme 1**, à chaque étape (ou temps t) on échantillonne parmi les données d'entraînement un *batch* aléatoire L_t (avec une probabilité de L/N, où L est la taille de ces groupes et N celle des données), puis dessus :

- Sur chacun des éléments x_i
  - On calcule le gradient de la fonction de perte $\mathbf{g}_t(x_i) = \nabla_{\theta_t}\mathcal{L}(\theta)$
  - On le remet à l'échelle si besoin, de manière à limiter sa norme euclidienne ℓ_2 au seuil maximal C fixé (c'est l'un des hyperparamètres). Notons $\bar{\mathbf{g}}_t(x_i)$ le résultat. Remarquons que si cette étape d'écrêtage ou *clipping* est classique en SGD standard, elle s'exerce alors généralement sur la valeur moyenne calculée ci-après, plutôt que sur chacune des données comme ici où l'on doit

limiter leur influence individuelle pour pouvoir garantir la confidentialité différentielle.

  - On ajoute un bruit aléatoire qui suit la loi normale centrée d'écart-type σC suivant chaque dimension, soit $\mathcal{N}(0, \sigma^2 C^2 \mathbf{I})$. La justification de la DP s'appuie sur la majoration de la norme ℓ_2 du gradient garantie par l'étape prédédente.

- On calcule la moyenne $\tilde{\mathbf{g}}$ de ces gradients individuels bruités. **pas clair : bruit sur chaque ?**

- On ajuste les paramètres comme pour une SGD classique, compte tenu du taux d'apprentissage $\eta_t > 0$, autrement dit : θ_{t+1} prend la valeur $\theta_t - \eta_t \tilde{\mathbf{g}}_t$.

Dans le cas d'un réseau multi-couches, chacune peut être traitée individuellement, avec ses propres valeurs de C et de σ. De plus, ces dernières au temps $t+1$ peuvent varier en fonction de l'étape t (pour les exemples donnés, on restera cependant sur des réglages constants pour C et σ).

Une distinction peut être faite entre un ***lot*** (*in english*), groupe de taille L de données utilisées pour calculer la moyenne empirique du gradient et le ***batch*** correspondant au groupement habituel pour le calcul de gradient. Ainsi, pour limiter la consommation de mémoire, on pourrait choisir une taille de *batch* bien inférieure à L, sur lequels on effectuerait les calculs, et ajouter le bruit sur le lot constitué de l'ensemble des *batches*. En pratique, par souci d'efficacité on permute aléatoirement les entrées, puis on partitionne. Mais pour les raisonnements, on considère que chaque exemple est tiré avec une probabilité de $q = L/N$, N étant la taille du *dataset* d'entraînement.

2.2 Comptabilité du budget de confidentialité plus en détails

La compatibilité avec la composition permet les calculs nécessaires : à chaque utilisation de données d'entraînement, on en consomme une partie qui se cumule avec les autres « dépenses ». En choisissant $\sigma = \sqrt{2 \log \frac{1{,}25}{\delta}}/\varepsilon$, chaque étape est (ε, δ)-DP vis à vis du *lot*, lui-même échantillonné sur la base complète, donc $(O(q\varepsilon), q\delta)$-DP par rapport à l'ensemble des données d'entraînement, avec $\varepsilon < 1$ (*cf. théorème d'amplification, Kasiviswanathan* 2 011). configure Pour améliorer le résultat obtenu avec le théorème fort de composition, meilleur résultat jusque là, mais qui ne tient pas compte la distribution du bruit ajouté, la nouveauté est l'utilisation du ***moments accountant*** dans de cadre de la DP-SGD (et plus simplement la SGD standard). On prouve qu'en choisissant convenablement le niveau de bruit et le seuil de *clipping*, la DP-SGD est $(O(q\varepsilon\sqrt{T}), \delta))$-DP où T est le nombre de cycles, d'*epoch*. Le gain est d'un facteur $\sqrt{log(1/\delta)}$ sur ε et de Tq sur δ par rapport au théorème fort de composition.

Au final, le **théorème 1** montre qu'il existe des constantes c_1 et c_2 telles qu'étant donnés $q = L/N$ le taux d'échantillonnage et T le nombre d'étapes, l'algorithme précédent est (ε,δ)-DP pour tout $\varepsilon < c_1 q^2 T$ et tout $\delta > 0$, si l'on choisit

$$\sigma \geqslant c_2 \frac{q\sqrt{T\log(1/\delta)}}{\varepsilon}$$

Le gain théorique par rapport au théorème fort de composition est vérifié en pratique.

L'élément comptable *moments accountant* est conçu spécifiquement pour suivre le budget de confidentialité, de manière « serrée ». La perte de confidentialité ou ***privacy loss***, pour un mécanisme $\mathcal{M}$, une entrée auxilaire aux (qui sera typiquement constitudée des sorties aux étapes précédentes, dans les enchaînements d'appels), des bases adjacentes d et d' et une sortie o, est définie par

$$c(o;\mathcal{M},\text{aux},d,d') = \log \frac{\mathbb{P}[\mathcal{M}(\text{aux},d) = o]}{\mathbb{P}[\mathcal{M}(\text{aux},d') = o]}$$

On définit alors le **λ^{e} moment** $\alpha_{\mathcal{M}}(\lambda;\text{aux},d,d')$ comme le logarithme de la fonction génératrice des moments, en λ, soit $\log \mathbb{E}_{0\sim\mathcal{M}(\text{aux},d)}[\exp(\lambda\, c(o;\mathcal{M},\text{aux},d,d')]$ et on utilise

$$\alpha_{\mathcal{M}}(\lambda) = \max_{\text{aux},d,d'} \alpha_{\mathcal{M}}(\lambda;\text{aux},d,d')$$

(où la maximum est pris sur toutes les entrées auxiliaures aux et toute les bases adjacentes d et d') pour la garantie de confidentialité. Le **théorème 2** montre alors que si l'on enchaîne une séquence (« adaptative », où chaque phase peut utiliser sur les sorties des précédentes) de mécanismes $\mathcal{M}_i$, on a

$$\alpha_{\mathcal{M}}(\lambda) \leqslant \sum \alpha_{\mathcal{M}_i}(\lambda)$$

(compatibilité avec la composition)

et pour tout $\varepsilon > 0$, le mécanisme $\mathcal{M}$ est (ε,δ)-DP pour

$$\delta = \min_{\lambda}\ \exp(\alpha_{\mathcal{M}}(\lambda) - \lambda\varepsilon)$$

(majoration de la queue de la distribution)

Il suffit donc de majorer $\alpha_{\mathcal{M}_i}(\lambda)$ à chaque étape* pour borner le moment global, puis d'utiliser la seconde partie du théorème pour convertir cela en garantie de (ε,δ)-DP. La principale difficulté reste au niveau de la première phase (*cf.* *). Dans le cas du mécanisme gaussien avec échantillonnage aléatoire de probabilité q, on montre que

$$\alpha(\lambda) \leqslant q^2\lambda(\lambda+1)/(1-q)\lambda^2 + O(q^3/\sigma^3)$$

2.3 Réglage des hyper-paramètres

La justesse (*accuracy*) du modèle s'avère plus sensible aux paramètres d'entraînement comme la taille du *batch* et le niveau de bruit qu'à la structure du réseau. En particulier pour les fonctions d'objectif non convexes, il est préférable de choisir une taille de *batch* supérieure à 1, contrairement aux cas convexes, sans toutefois être trop importante pour préserver la confidentialité (*cf.* théorème 1), disons du même ordre de grandeur que le nombre attendu d'*epochs*.

En DP-SGD, le taux d'apprentissage n'aura pas à être progressivement diminué jusqu'à une très faible valeur. On constate qu'on a intérêt à partir d'un niveau assez grand, qui diminue linéairement sur quelques *epochs*, puis reste constant ensuite.

3 Expérimentations

[*Note : comme c'est TensoFlow qui est utilisé ici, on détaillera moins, tâchant plutôt de repérer ce qui pourrait avoir été repris dans pytorch-dp..*]

On retrouve les phases de *sanitization* (*clipping*) du gradient, puis d'ajout de bruit avec suivi du budget de confidentialité alloué. Pour la première, on retrouve le même module `per_example_gradient` que dans pytorch, qui effectue ce calcul efficacement (sur des couches cachées classiques, mais pas à ce stade sur celles de convolution
à comparer donc avec « notre » solution qui devrait les prendre en charge...)

Le *moments accountant* est utilisé pour le second point. On peut traduire à tout moment le résultat en (ε,δ)-DP, plus facilement interprétable. Le $\alpha(\lambda)$ est calculé à partir de μ_0 et μ_1, les densités des lois normales centrées respectivement en 0 et1 et d'écart-type σ. On pose $\mu = (1-q)\mu_0 + q\mu_1$ et $\alpha(\lambda)$ vaut $\log\max(E_1,E_2)$ où $E_1 = \mathbb{E}_{z\sim\mu_0}[(\mu_0(z)/\mu(z))^\lambda]$ et $E_2 = \mathbb{E}_{z\sim\mu}[(\mu(z)/\mu_0(z))^\lambda]$. Pour éviter de faciliter une attaque basée sur l'analyse de l'adaptation dynamique des paramètres, ces derniers sont ici fixés *a priori*.

3.1 DP-PCA

Un prétraitement des données par analyse en composantes principales (PCA) est intéressante pour optimiser l'utilisabilité des données d'entraînement. Pour la rendre DP-compatible, ici, une technique connue est reprise (échantillonnage aléatoire, transformation en vecteurs-lignes ℓ_2-normalisés pour former une matrice A, puis ajout de bruit gaussien à la matrice de covariance A^TA avant la PCA sur cette dernière). Cela consomme une part du budget de confidentialité, mais semble intéressant étant donné le gain de qualité sur le modèle et la réduction du temps d'entraînement induite.

4 Expérimentations

L'idée est notamment de comparer la consommation du budget de confidentialité obtenue, par rapport à l'application du théorème for de composition. Les paramètres sont calculé en fonction de

- σ le niveau de bruit.
- $q = L/N$ le ratio d'échantillonnage : une *epoch* consiste donc en $1/q$ *batches*.
- E le nombre d'*epochs* (d'où E/q étapes).

On fixe $\delta = 10^{-5}$, et on calcule ε en fonction de E, avec $q = 0{,}01$ et $\sigma = 4$. Grossièrement, le gain est de l'ordre de 10 sur ε pour quelques centaines d'*epochs*.

4.1 MNIST

Le réseau neuronal est composé d'une PCA en 60 dimensions (on note σ_p le bruit associé pour la DP), suivi d'une couche cachée unique à 1 000 unités avec ReLU (bruit *sigma* ; les résultats sont meilleurs qu'avec 2 couches cachées), en utilisant une taille des *lots* de 600, Le *clipping* du gradient est effectué à la norme 4 pour chaque couche. [*L'architecture choisie est donc assez différente de celle de l'exemple MNIST choisi pour pytorch-dp*].

Trois niveaux de bruit sont comparés, (σ,σ_p) vallant $(2; 4)$, $(4; 7)$ puis $(8; 16)$.

Le taux d'apprentissage est initialisé à 0,1. Il décropoît linéairement jusqu'à 0,052 sur 10 *epochs* puis reste constant.

4.2 CIFAR-10

-