Découverte de l'Apprentissage Profond
Application au Traitement du Langage Naturel

Radu CRACIUN

Mai 2019

Toute connaissance dégénère en probabilité.

DAVID HUME

Table des matières

Introduction

Pour l'heure, les technologies liées à l'intelligence artificielle et à l'apprentissage automatique ne sont que peu connues du grand public, contrairement à des outils tels que la messagerie ou la navigation chiffrées par exemple.

On peut distinguer au moins trois approches à l'apprentissage automatique et à ses utilisations : du point de vue du chercheur, c'est une théorie solide et complexe, fondée sur des mathématiques de haut niveau ; du point de vue de l'étudiant, il s'agit surtout de statistiques couplées à de l'algorithmique ; pour le profane, c'est simplement de la magie. En fait, on pourrait dire que l'aspect philosophique des problématiques liées à l'intelligence artificielle a été mieux compris que l'aspect technique. Certaines attentes ou craintes sont tout à fait fondées, d'autres, beaucoup moins. Une chose est sûre, l'allure à laquelle progressent les technologies de l'apprentissage rendent caduques l'immense majorité des prédictions, même parmi les experts.

Comment peut-on envisager qu'une machine puisse reconnaître des visages, ou mieux encore, en générer, alors que personne ne lui a explicitement inculqué le concept de visage ? Que veut dire créer de l'art ou imiter un artiste ? Et, sans chercher l'alarmisme, est-il possible qu'un jour nous n'ayons plus d'emploi à cause de l'intelligence artificielle ? Voilà le type de questions qu'on ne peut s'empêcher de se poser, que l'on soit novice ou aguerri en la matière.

Nous adopterons ici le point de vue de l'étudiant et nous intéresserons à l'apprentissage artificiel en général, sans expliciter tous les détails, tout en essayant de mettre en pratique certaines notions. Plus précisément, nous avons souhaité découvrir l'apprentissage profond ainsi que certaines de ses applications au traitement de textes.

Le chapitre "Panorama de l'apprentissage artificiel" présente les notions indispensables de la théorie de l'apprentissage. Par la suite, nous nous concentrerons sur les réseaux de neurones profonds dans le chapitre "Entraînement et évaluation d'un réseau de neurones". Dans ces premiers chapitres, nous resterons très théoriques, tout en présentant des exemples simples. A partir du chapitre "Traitement du langage naturel", nous présenterons une implémentation possible d'un classifieur et d'un générateur, tous deux fonctionnant grâce à des réseaux de neurones profonds. On s'appuiera pour cela sur un corpus de textes français classiques.

Chapitre 1

Panorama de l'apprentissage artificiel

Cette partie est destinée à présenter les notions incontournables de l'apprentissage artificiel, sans se prétendre exhaustive ni très détaillée. On y traite de la régression, de la classification, de l'apprentissage supervisé et non supervisé, et enfin des neurones artificiels. Ce panorama est construit par niveau croissant de difficulté des notions, et chaque section fait appel aux précédentes.

1.1 Régression *versus* classification

1.1.1 Régression

La ***régression*** se caractérise par deux aspects : les entrées de la méthode sont des valeurs numériques continues et la sortie est une autre variable continue. L'idée est contenue dans le nom, c'est-à-dire qu'à partir des diverses mesures effectuées, on tente de tout résumer pour en déduire une autre valeur [1].

Un exemple typique est celui d'une maison dont on cherche à estimer le prix. Si l'on considère uniquement sa surface habitable, on pourra éventuellement en prédire le prix, mais il faut s'attendre à se tromper assez lourdement. En revanche, si l'on prend en considération l'âge,la durée depuis la mise en vente, l'efficacité thermique, le prix moyen des maisons environnantes, on peut affiner notre estimation. Pour prédire un tel prix, il suffit de construire un modèle, de le tester sur un grand jeu de données réelles puis de l'ajuster. Une difficulté est que, selon le modèle qu'on prend, ses paramètres ne sont pas forcément uniques. On peut donc être amené à tester un grand nombre de paramètres mais aussi un grand nombre de modèles si l'on souhaite ouvrir une agence immobilière et fixer des prix.

Le modèle de régression le plus connu est la ***régression linéaire***. Ses hypothèses sont simples : la variable prédite est une combinaison linéaire des variables d'entrées. Dans le cas de notre exemple, si nous appelons $\hat{y}$ le prix de la maison, x_i les variables d'entrée et α_i les paramètres du modèle, alors la prédiction s'écrit :

$$\hat{y} = \alpha_0 + \sum_{i=1}^{i=n} \alpha_i x_i$$

ou encore

$$\hat{y} = \langle \boldsymbol{\alpha}, \boldsymbol{x} \rangle \tag{1.1}$$

en notation vectorielle, où : $\boldsymbol{\alpha}$ est le vecteur des $n+1$ paramètres ; $\boldsymbol{x}$ est le vecteur des paramètres, de dimension $n+1$, dont le premier vaut 1 ; $\langle , \rangle$, représente le produit scalaire.

Ainsi, avec un seul paramètre, par exemple la surface habitable, l'évolution du prix pourrait ressembler à ceci :

[FIGURE] où chaque point représente une maison dont on a mesuré la surface habitable et consulté le prix de vente.

Ce modèle est souvent privilégié du fait de la simplicité de sa mise en œuvre. Bien sûr, on peut imaginer des modèles non linéaires, dans lesquels $\hat{y}$ est une fonction non linéaire des x_i. Quelle que soit

1. Ici, les variables continues n'évoluent pas dans un ensemble continu de valeurs tels que décrits par la théorie des ensembles. En fait, on pourrait se contenter de savoir s'il existe une relation d'ordre entre les valeurs, et si ces dernières sont nombreuses, pour conclure que la variable est continue.

la régression choisie, il existe une méthode suffisamment générale pour calculer une approximation des meilleurs paramètres du modèle, nous la verrons plus loin.

Le choix du modèle se fait par souci de ***généralisation***, c'est-à-dire qu'un modèle sera privilégié s'il permet de prédire des prix réalistes au-delà du seul jeu de données sur lequel il a été construit. Là encore, nous verrons plus loin comment parvenir à un modèle satisfaisant dans le cas général.

Enfin, il est possible de faire une régression sur des variables catégorielles, mais il faut au préalable les "numériser". Cette méthode est appelée "analyse factorielle des correspondances", mais les détails importent peu ici.

Voyons maintenant un autre pilier sur lequel peut s'appuyer un algorithme d'apprentissage pour fonctionner.

1.1.2 Classification

Contrairement à la régression, la ***classification*** (***clustering*** en anglais) prédit la valeur d'une variable discrète, appelée ***classe***, à partir d'entrées continues ou discrètes.

Pour reprendre l'exemple précédent, il pourrait s'agir de prédire si la maison serait éligible au classement des monuments historiques. Dans le premier cas on parle de "classification binaire". Si plus de 3 valeurs sont possibles, on parle de "classification multiple" ou "multiclasses". Parfois, il arrive que des classes ne soient pas mutuellement exclusives, mais nous ne nous en préoccuperons pas ici.

Les modèles de classification sont nombreux et très divers. Ceci est dû au fait que les problèmes auxquels répondent les algorithmes de classification sont eux-mêmes très divers. Pour les modèles, on citera donc pêle-mêle les arbres de décision (et leur amélioration que sont les forêts aléatoires), les k-plus proches voisins, les k-moyennes, ou encore la classification hiérarchique.

[SCHEMA]

Le choix entre ces modèles se fait souvent en fonction de la dimension du problème, mais aussi de la répartition des données dans l'espace des variables. Typiquement, si les données peuvent être séparées par une droite, un plan, ou un hyperplan, alors on se contentera d'un classifieur linéaire adapté à la dimension. Mais il arrive bien souvent que les données ne sont pas séparables linéairement, alors on doit pousser l'analyse plus loin, comme leur appliquer des transformations non linéaires ou les traiter avec des réseaux de neurones.

Conclusion

Nous avons vu que l'apprentissage artificiel se réduisait à résoudre un problème de type régression, calcul d'une valeur continue, ou bien classification, calcul d'une valeur discrète.

Il s'avère que certaines techniques de régression permettent aussi de faire de la classification, typiquement en calculant une probabilité qu'on arrondit ensuite au nombre entier le plus proche (0 ou 1). C'est le cas de la régression ridge par exemple. Nous n'en donnerons pas les détails ici.

Abordons maintenant une autre dichotomie classique du domaine de l'apprentissage.

1.2 Apprentissage supervisé *versus* non supervisé

1.2.1 Supervisé

L'***apprentissage supervisé*** consiste simplement à entraîner l'algorithme avec des exemples. On lui soumet un ensemble de données dont on donne le label ou la valeur qu'il devrait prédire et on ajuste les paramètres du modèle pour "coller" à ces exemples. Ainsi, on donnerait 500 instances de maisons et de leurs caractéristiques, ainsi que leur prix (pour la régression) ou le label "bâtiment classé"/"maison ordinaire" (pour la classification).

Plus précisément, on commencera par définir une ***fonction de coût*** (***loss function*** en anglais) dépendant de la réponse attendue et de la réponse réellement produite par le modèle ; il s'agit en fait d'une mesure de la performance du modèle sur les données d'entraînement. Pour une régression linéaire, on choisit très souvent la fonction racine de l'erreur quadratique moyenne $\sqrt{\frac{1}{m}\sum_1^m(\hat{y}_i - y_i)^2}$, mais d'autres sont possibles. Le but de l'algorithme sera alors de trouver les paramètres du modèle qui

minimisent la fonction de coût. Dans le cas particulier d'une régression linéaire, il existe une forme close pour calculer ces paramètres optimaux (lorsque la fonction de coût est la racine de l'erreur quadratique moyenne). Mais dans le cas général, nous ne connaissons pas de formule. Pour trouver les paramètres induisant une bonne performance sur le jeu d'entraînement, on utilise une heuristique (voir la section 2.1.2).

Une fois l'algorithme entraîné, donc une fois les paramètres du modèle fixés, on peut lui soumettre des données de test pour voir s'il est capable de généraliser et jusqu'à quel point. Si ces données sont annotées comme lors de la phase d'entraînement, on pourra évaluer le pouvoir de généralisation de l'algorithme, à nouveau grâce à la fonction de perte. Si on n'annote pas ces données de test, on aura simplement une vague idée du comportement sous-jacent de l'algorithme [2].

1.2.2 Non supervisé

La caractéristique commune à toutes les méthodes d'***apprentissage non supervisé*** est l'absence de phase d'entraînement par l'exemple, contrairement à l'apprentissage supervisé. La motivation première pour employer de telles techniques est de mettre en lumière des structures sous-jacentes des données, ce qui s'apparente donc à des statistiques exploratoires multidimensionnelles. Diverses méthodes peuvent s'apparenter à de l'apprentissage non supervisé, comme la réduction de dimmensionnalité (notamment par l'analyse en composantes principales) ou la détection d'anomalies (avec les réseaux bayésiens par exemple). Ici encore, ces méthodes peuvent être réduites à une classification (garder ou non une variable, considérer une observation comme normale ou anormale), ou à une régression (distance moyenne entre deux points de l'espace des variables) ; ou encore, une régression dont on déduit une classe.

Malheureusement, l'apprentissage non supervisé le plus général dépasse largement le cadre de ce projet. Continuons donc sur les tant plébiscités neurones artificiels.

1.3 Notion de neurone artificiel

Les sciences s'inspirent parfois de caractères trouvés dans la nature et essaye de les imiter. Mais l'imitation ne réussit pas toujours aussi bien que l'original. Si l'on s'était profondément inspiré des oiseaux pour voler, nos avions battraient des ailes. Or, laisser de côté le biomimétisme a permis le développement des turboréacteurs nous permettant de nous déplacer quasiment à la vitesse du son lors d'un voyage de routine.

Pour les neurones, l'histoire est sensiblement la même. Les neurones biologiques sont des cellules fonctionnant en réseau et dont un des rôles est de transmettre des impulsions électriques. L'interconnexion entre ces neurones se fait par voie chimique, et leur agencement change au cours de la vie.

[COMPLETER]

1.3.1 Unité Linéaire à Seuil

L'Unité Linéaire à Seuil (ULS, ou Linear Threshold Unit, LTU en anglais) est un composant numérique qui à plusieurs entrées pondérées associe une sortie binaire. Après avoir calculé une combinaison linéaire des entrées, il applique une fonction seuil. Sa sortie vaut :

$$o = s(\langle \boldsymbol{w}, \boldsymbol{x} \rangle)$$

où : s est la fonction seuil ; $\boldsymbol{w}$ est le vecteur des n poids associés aux entrées ; $\boldsymbol{x}$ est le vecteur des n entrées.

2. Ces dernières années, la classification de photos et d'images a connu un essor impressionnant grâce à des architectures complexes de réseaux de neurones, et les performances des algorithmes frôlent le sans-faute très souvent. Il existe cependant une manière particulière de tester les algorithmes, volontairement malicieuse : il s'agit de "l'attaque à 1 pixel" (ou sa généralisation, "construction adversariale"). Il s'agit de modifier un unique pixel, bien choisi, d'une image, de sorte que l'algorithme se trompe dans sa prédiction. Sur une image de 32x32 pixels 3 canaux, cela représente donc une perturbation de moins de 0.01% des données. De manière similaire, on peut modifier de multiples pixels d'une image, sans que ce soit visible à l'œil nu et de sorte à tromper l'algorithme. Dans ces deux cas, la question de la sécurité offerte par ces algorithmes est centrale, et de nombreuses recherches sont en cours.

Un LTU a son utilité dans une classification binaire lorsqu'il est utilisé seul, ou dans une classification multiple lorsqu'il est combiné à d'autres LTU. Cependant, tel quel, il est incapable de séparer des points autrement que par une frontière linéaire. Par exemple, simplement pour mimer le comportement de la fonction OU EXCLUSIF, il faut ruser et combiner plusieurs LTU :

[SCHEMA]

[DETAIL]

L'apparente faiblesse des LTU face à un problème si simple a quelque peu déçu les chercheurs des années 50-60, qui ont alors pu abandonner le connexionisme. Mais nous verrons que les LTU seuls ou en réseau sont loin d'être inutiles.

Voyons maintenant quelles modifications ont été apportées depuis.

1.3.2 Neurone non linéaire

Un désavantage majeur des LTU est que leur sortie, binaire, est calculée par une fonction seuil. Ceci mène à une frontière de séparation linéaire dans l'espace des variables d'entrée. Bien que la simplicité des modèles linéaires ait fait ses preuves par le passé, il est parfois nécessaire de complexifier. Ainsi, une modification envisagée fut de remplacer la fonction seuil par une des fonctions suivantes (ou par une des nombreuses autres existantes) :

$$\sigma : t \mapsto \frac{1}{1+e^{-t}} \tag{1.2}$$

$$\tanh : t \mapsto \frac{e^{t}-e^{-t}}{e^{t}+e^{-t}} \tag{1.3}$$

$$ReLU : t \mapsto max(0,t) \tag{1.4}$$

Voici les représentations graphiques des fonctions et de leurs dérivées autour de $(0,0)$:

[FIGURE]

Un autre ajout fut celui d'un terme constant, appelé ***biais*** avant la ***fonction d'activation***.

Voici un schéma représentant ce nouveau type de neurone :

[SCHEMA]

En conservant les mêmes notations que précédemment, l'équation en sortie du neurone est donc la suivante :

$$o = f(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b) \tag{1.5}$$

Le biais du neurone sert à activer celui-ci même lorsque le signal d'entrée est faible. Dans un réseau, cela aura pour effet de déclencher l'apprentissage chez au moins quelques neurones.

L'intérêt des neurones non linéaires apparaît lorsqu'on les regroupe en réseaux.

1.4 Réseaux de neurones

Voici un aperçu de ce à quoi un réseaux de neurones peut ressembler :

[SCHEMA]

On y voit une couche d'entrée, des couches cachées et une couche de sortie. Mais, contrairement au neurone seul, on ne peut pas simplement expliciter la sortie du réseau en fonction des entrées. Nous expliquerons en détail aux chapitres suivants comment travailler avec de tels réseaux.

1.5 Dilemme biais - variance, ajustement

Imaginons la situation suivante : deux amis, Béatrice et Valentin, se baladent et croisent un artiste de rue. Celui-ci leur présente une pièce de monnaie et les assure qu'elle n'est pas truquée. Sans tarder, il leur propose de deviner de quel côté tombera la pièce au 9e lancer, après avoir regardé les 8 premiers lancers. Les deux amis constatent qu'à chacun des 8 lancers, la pièce est tombée sur pile, mais interprètent les résultats différemment. Pour Béatrice, chaque lancer est indépendant du précédent, et on

leur a de plus dit que la pièce n'était pas truquée ; elle en conclut que les deux faces sont équiprobables au 9e lancer. Valentin n'est pas d'accord. Selon lui, on ne peut pas faire confiance à ce qu'a dit l'artiste de rue au début, les données "parlent" d'elles-mêmes ; il en conclut que la pièce tombera sur pile.

Malheureusement, l'histoire ne dit précise pas la suite des évènements, on peut malgré tout en tirer des leçons. Le raisonnement de Valentin souffre d'un défaut relativement facile à déceler, appelé ***sur-ajustement*** (on parle aussi de ***sur-interprétation*** ou de ***sur-apprentissage***, ***overfitting*** en anglais), qui survient lorsqu'on cherche à généraliser des données mais à partir d'un modèle qui les ajuste parfaitement. En résumé, Valentin a tendance à trop "coller" aux données, en leur faisant aveuglément confiance. Dans ce cas précis, on pressent qu'il tend à surajuster car, sous l'hypothèse que la pièce est équilibrée, 8 tirages identiques d'affilée surviennent tous les 256 lancer environ. Ainsi, Valentin n'a pas suffisamment de données pour écarter si drastiquement le modèle de Béatrice. En revanche, si nos deux amis avaient été témoins de plus de lancers identiques, disons 50, il aurait été bien plus légitime de conclure que le modèle d'équiprobabilité est faux.

Malgré tout, son amie n'a pas tout à fait raison non plus. L'approche de Béatrice se fonde sur un modèle très simplifié de la réalité, et les prémisses de celui-ci peuvent être fausses. Dans le cas d'une pièce de monnaie, il est possible de s'entraîner à la lancer et la rattraper de sorte à en contrôler l'issue avec une forte probabilité. Des études spécifiques on été menées sur des pièces de 1 euro et 2 euros, concluant qu'elles n'étaient pas équilibrées [3]. Dans le cas de Béatrice, lorsqu'on fait trop confiance à un modèle et pas assez aux données, on parle de ***sous-ajustement*** (***sous-apprentissage***, ou encore de ***sous-interprétation*** , ***underfitting*** en anglais).

Cet exemple élémentaire sert à illustrer un problème central de l'apprentissage artificiel : le dilemme entre diminuer le biais [4] et diminuer la variance. Un résultat important en statistiques et apprentissage artificiel est que l'erreur de généralisation d'un modèle possède trois sources d'erreur différentes :

Biais Il est dû à des mauvaises hypothèses, comme par exemple considérer que des données obéissent à un modèle linéaire alors qu'un processus quadratique en est réellement à l'origine. Un modèle à haut biais a tendance à sous-ajuster les données.

Variance Elle résulte d'une sensibilité excessive du modèle vis-à-vis des données. C'est typiquement le cas des modèles de régression polynomiaux à degré élevé. Un modèle à haute variance a tendance à sur-ajuster les données.

Erreur résiduelle Cette dernière forme d'erreur est ce qu'on appelle parfois le "bruit" qui s'immisce dans les données. L'unique moyen de s'en débarrasser est de réparer les sources de données, comme des capteurs défaillants par exemple, ou d'ignorer les données aberrantes. Cette tâche est loin d'être aisée dans le cas général.

Tenter de réduire le biais d'un modèle revient à en augmenter la variance, et vice versa, d'où le terme de "dilemme" ou "compromis" biais-variance.

Nous décrirons plus loin comment déceler la sur-interprétation et la sous-interprétation, ainsi que diverses techniques pour les éviter.

1.6 Protocole pour un problème d'apprentissage

Une partie importante du travail est faite lors de l'analyse du problème à résoudre. Voici une approche assez générale pour clarifier les points précédents :

Données à traiter Est-ce du texte, des images, du son, de la vidéo ?

Format des données S'agit-il des fichiers colonnes à en-têtes, du texte brut, des images (éventuellement compressées), du son très riche ?

Sortie de l'algorithme L'algorithme doit-il prédire un mot, une phrase, une valeur numérique, une couleur, une image ?

Structures des observations Les observations sont-elles distribuées de manière normale, bimodale, par petits groupes, sur une droite, sur un hyperplan ?

3. vidéo
4. Ce biais, notion statistique, n'a rien à voir avec celui des neurones artificiels.

Modèle adapté L'état de l'art suggère-t-il une régression linéaire, une régression polynomiale, un classifieur linéaire, un réseau de neurones ?

Ne pas répondre (correctement) à ces questions avant de se lancer dans la programmation est une erreur souvent fatale.

1.7 Conclusion

Nous avons ici succinctement présenté quelques manières de catégoriser les solutions à un problème d'apprentissage automatique, de la régression linéaire à l'apprentissage non supervisé et aux réseaux de neurones profonds. Enfin, nous avons vu un rapide protocole nous permettant de décider quelle solution envisager d'abord lorsqu'on doit résoudre un problème d'apprentissage ou d'intelligence artificielle.

Malheureusement, il n'est pas possible de rédiger un état de l'art des techniques et des technologies de ce domaine sans écrire un (très gros) livre. Cependant, survoler les méthodes tout en mettant l'accent sur les principes reste intéressant.

Nous verrons, au prochain chapitre, comment entraîner et évaluer un réseau de neurones, et les problématiques qui émergent lorsque la taille du réseau augmente.

Chapitre 2

Entraînement et évaluation d'un réseau de neurones

Dans ce chapitre, nous traiterons des moyens généraux d'entraîner un réseau de neurones de manière supervisée, ainsi que des méthodes de l'évaluer en vue de l'améliorer.

2.1 Entraînement

L'***entraînement*** d'un réseau de neurones, est une phase durant laquelle on force le réseau à changer les poids et biais de ses neurones pour produire la sortie voulue pour des exemples connus ; on espère ainsi que le réseau sera capable de produire la bonne sortie pour des instances encore inconnues, c'est-à-dire bien généraliser.

2.1.1 Fonction objectif

Une manière classique de suivre l'évolution du système qui apprend est de calculer un indicateur de sa performance. En l'occurrence, il s'agit d'évaluer sa performance sur le jeu de données qui lui est soumis lors de la phase d'entraînement. Le moyen le plus répandu est de calculer cet indicateur sur chacune des instances d'entraînement x_i et d'en prendre la moyenne sur l'ensemble des m instances. On appelle cet indicateur ***fonction de coût***, ***fonction de perte*** ou encore ***fonction objectif*** (***cost function*** ou ***loss function*** en anglais). Pour un réseau de neurones, on définit souvent une des fonctions de coût suivantes :

$$\kappa = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{2.1}$$

$$\kappa = -\frac{1}{m}\sum_{i=1}^{m} y_i \ln \hat{y}_i + (1 - y_i)\ln(1 - \hat{y}_i) \tag{2.2}$$

où : m est le nombre d'observations ; les y_i sont les réponses attendues et $\hat{y}_i$ sont les réponses inférées par le réseau.

Maximiser la performance du réseau est équivalent à minimiser la fonction de coût, mais le formuler ainsi permet de profiter des techniques d'optimisation déjà existantes. On en traite dans la section suivante.

La dépendance entre la fonction de coût et les réponses y_i et $\hat{y}_i$ est implicite. En effet, on ne peut pas les modifier directement. D'une part, y n'est pas un paramètre du modèle mais une caractéristique des données, et d'autre part, $\hat{y}$ est issu d'un calcul complexe effectué à partir des données d'entrée ainsi que des poids et biais du réseau de neurones. Il serait donc plus judicieux d'expliciter la relation entre paramètres du modèle, les valeurs d'entrée et la sortie qu'il produit :

$$\kappa = \kappa(\boldsymbol{w}, \boldsymbol{b}, \boldsymbol{x}, \boldsymbol{y})$$

où les variables sont les regroupements des poids (respectivement des biais, entrées et réponses attendues) sous forme de vecteurs. Cette représentation est utile pour l'algorithme qui suit.

[CORRIGER]

[FINIR]

2.1.2 Descente de gradient

La ***descente de gradient*** est une méthode d'optimisation très générale. Elle permet de minimiser une fonction, donc de maximiser son opposé, par étapes successives. Un point caractéristique de la méthode est qu'elle ne garantit, à condition que la fonction soit bien choisie, que la découverte d'un extremum local. De manière générale, la découverte du extremum global d'une fonction continue, s'il existe, est un problème difficile.

Bien évidemment, puisqu'il s'agit de minimiser la fonction de coût, ses propriétés vont grandement influer sur le déroulement de l'algorithme. Le cas le plus intéressant est celui d'une fonction convexe [DEFINIR], ce qui nous assurerait que la fonction possède au plus un minimum. Si ce n'était pas le cas, la descente de gradient pourrait s'arrêter dans un minimum local loin de l'optimum global. Mais il n'est pas toujours évident de garantir cette propriété, encore moins en dimension 2 ou supérieure.

L'idée de l'algorithme de la descente de gradient est qu'à chaque itération on ajuste les paramètres de la fonction de coût, de sorte que les nouvelles valeurs renvoient une sortie plus faible qu'à l'itération précédente. On choisit de s'arrêter lorsque les variations inter-étapes sont plus faibles qu'un seuil choisi, ou lorsqu'on a effectué un certain nombre d'itérations.

Nous avons évoqué les fonctions convexes, mais une condition nécessaire est en fait la différentiabilité de la fonction de coût. Lorsque c'est le cas, on peut associer un gradient à chaque point de cette fonction. Le gradient est un vecteur dont le sens indique la plus forte pente au voisinage du point où il est calculé. Ses coordonnées, en même nombre que les arguments de la fonction de coût, donnent directement la marche à suivre : leur signe indique comment faire varier les paramètres, et leur valeur indique de combien les changer.

L'algorithme, incroyablement simple, est le suivant :

Algorithm 2.1 Descente de gradient

$\boldsymbol{x} \leftarrow rand()$ ▷ Initialisation aléatoire
repeat
  $\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \cdot \boldsymbol{\nabla}\kappa$
until $\|\boldsymbol{\nabla}\kappa\| \leq \varepsilon$

où : η est un facteur appelé ***taux d'apprentissage***. On parle d'***hyperparamètre*** dans ce cas, car il n'est pas mis à jour durant l'apprentissage mais durant une phase ultérieure [1].

En une dimension, cela pourrait se représenter comme suit :

[SCHEMA]

Avec deux paramètres, on pourrait l'imaginer comme ceci :

[SCHEMA]

[COMPLETER]

[COMPLETER : SGD et pourquoi]

2.1.3 Rétropropagation de gradient

La ***rétropropagation de gradient***, (***gradient backpropagation*** en anglais) redonna un souffle de vie aux réseaux de neurones, lesquels résistaient à toutes les tentatives pour les entraîner convenablement. Son principe est très simple, même si la mise en œuvre requiert un peu de réflexion.

L'idée est la suivante : dans un réseau de neurones, chaque neurone prend en entrée la sortie d'autres multiples neurones, donc sa propre sortie en dépend. Ainsi, toute erreur de sortie du réseau est imputable non seulement à la couche de sortie mais à tous les neurones du réseau. Cependant,

1. En réalité, il pourrait être modifié durant l'apprentissage, selon certaines règles plus ou moins complexes. Nous en traitons dans la section "Validation et ajustement d'hyperparamètres".

la responsabilité dans cette erreur n'est pas équitablement partagée. Il faut alors déterminer assez précisément la contribution de chaque neurone à la réponse obtenue en sortie, et propager le résultat vers l'"arrière", à travers les couches de neurones, c'est-à-dire vers la couche d'entrée.

Il est donc clair que la sortie o d'un neurone est fonction implicite des nombreux paramètres précédents. Donc La variation de l'entrée par rapport à l'un de ces paramètres w se calcule comme suit :

$$\frac{\partial o}{\partial w} = \frac{\partial o}{\partial \alpha}\frac{\partial \alpha}{\partial w} \tag{2.3}$$

où α est un autre paramètre quelconque dont dépend o.

L'intérêt de l'expliciter, et notamment en insérant tous les paramètres intermédiaires, est que les calculs sont alors immédiats : la variation de la sortie de chaque neurone est directement calculée par rapport à la variation des paramètres de ce même neurone.

Considérons deux neurones, à une seule entrée, le second étant connecté à la sortie du premier. La règle explicitée ci-dessus stipule que la variation de la sortie $\hat{y} = o_2 = f(w_2 o_1 + b_2) = f(z_2)$ (on rappelle que f est la fonction d'activation du neurone) par rapport à celle du poids w_1 vaut :

$$\begin{aligned}\frac{\partial o_2}{\partial w_1} &= \frac{\partial o_2}{\partial z_2}\frac{\partial z_2}{\partial o_1}\frac{\partial o_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} \\ &= f'(z_2) \times w_2 \times f'(z_1) \times x_1\end{aligned}$$

De même, la variation de la sortie par rapport au biais du premier neurone vaut :

$$\begin{aligned}\frac{\partial o_2}{\partial b_1} &= \frac{\partial o_2}{\partial z_2}\frac{\partial z_2}{\partial_1}\frac{\partial o_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} \\ &= f'(z_2) \times w_2 \times f'(z_1)\end{aligned}$$

Finalement, on pourrait presque considérer que "rétropropagation" est simplement un autre terme pour désigner la dérivée des fonctions composées. Mais généralement, on considère que cela désigne le processus suivant, pour chaque observation d'un jeu de données :

1. Propagation avant : calcul de la sortie $\hat{y}$ grâce aux paramètres du réseau
2. Calcul du coût $\kappa(y, \hat{y})$
3. Rétropropagation de l'erreur et des contributions
4. Mise à jour des paramètres dans toutes les couches par une itération de la descente de gradient

Une variante appliquée en pratique calcule d'abord les erreurs pour tous les échantillons sans mettre à jour les poids (on additionne les erreurs) et lorsque l'ensemble des données est passé une fois dans le réseau, on applique la rétropropagation en utilisant l'erreur totale. Cette méthode s'appelle entraînement par ***mini-lots*** (***mini-batch training*** en anglais), et offre de meilleurs temps d'exécution.

2.2 Tests

La phase de tests est celle durant laquelle on effectue le moins de changements sur les paramètres du modèle ou les hyperparamètres. Toutefois, elle est cruciale pour déceler les défauts d'ajustement, à savoir le sous-apprentissage et le sur-apprentissage. Pour ce faire, on étudie les ***courbes d'apprentissage***, qui sont la représentation graphique de la fonction de coût sur les données d'entraînement et les données de test. Voici un exemple fictif de ces courbes :

[SCHEMA]

L'interprétation est parfois subtile mais se fait généralement comme suit. Supposons qu'à la fin de chaque phase, les deux valeurs se soient stabilisées. Si les valeurs sont toutes les deux "hautes", on est confronté à du sous-apprentissage : le modèle est peu adapté dans tous les cas. Si les deux valeurs sont

"éloignées" l'une de l'autre, c'est un signe de sur-apprentissage : le modèle fait bien sur les données d'entraînement mais généralise mal. Dans le cas où au moins une des valeurs n'est pas stabilisée, il "suffit" de rajouter des observations.

Pour la phase de tests, il est évident qu'on doit instaurer une métrique de performances du réseau de neurones, alors que cela n'était pas si flagrant pour la première phase. De manière assez étonnante, on peut utiliser plusieurs indicateurs, donc pas uniquement la fonction de coût, pour évaluer le réseau sur des données de test. En fait, il serait même assez mal venu de se contenter de la fonction de coût pour la phase de tests. Prenons l'exemple d'un réseau résolvant un problème de classification binaire, comme celui de savoir si un texte a été écrit par Balzac ou non. Alors, on aimerait connaître les taux suivants :

Vrais positifs Textes de Balzac reconnus comme tels.

Vrais négatifs Textes d'autres auteurs et catégorisés "autres"

Faux positifs Textes attribués à Balzac mais à tort

Faux négatifs Textes de Balzac non reconnus comme tels

La phase de tests est la plus courte en terme de temps machine, mais sans doute la plus riche en interprétations.

2.3 Validation et ajustement d'hyperparamètres

Une dernière étape pour construire un modèle satisfaisant serait de simplement en comparer plusieurs et de sélectionner le meilleur (selon la métrique de la phase de tests). On pourrait faire de petits ou bien de gros changements dans les hyperparamètres de chaque algorithme puis les tester sur des données différentes de celles d'entraînement, appelées jeu de ***validation***.

Pour ce faire, on pourrait diviser le jeu de données initial en 3 : entraînement (60%), tests (20%), validation (20%). Ceci est envisageable avec des jeux de données immenses, pour lesquels réserver 40% n'est pas délétère. Mais il existe un moyen élégant de s'en passer, appelé ***validation croisée***. Plutôt que de réserver des données uniquement pour la validation, on découpe les données en k morceaux. On choisit alors un modèle et des hyperparamètres de l'algorithme. Ensuite, on entraîne le modèle sur tous les morceaux sauf le premier ; ce dernier sert à évaluer le modèle. On réinitialise ce modèle (on en efface les paramètres appris), puis on l'entraîne sur tous les morceaux sauf le second, qui, encore une fois, sert à évaluer le modèle. En répétant ce processus pour les k morceaux, on peut obtenir tout autant d'évaluations pour une combinaison d'hyperparamètres ; avec la validation classique, on aurait testé les hyperparamètres une seule fois (plus exactement sur un seul jeu de données) avant d'en choisir d'autres.

Bien sûr, même si on a la possibilité de changer les valeurs des hyperparamètres, on ne sait pas nécessairement quelle valeur leur donner. On a développé des techniques pour ce cas, mais en principe, optimiser des hyperparamètres est un problème d'optimisation classique. On citera donc comme techniques la recherche aléatoire, la recherche sur grille, l'optimisation évolutionniste et même la descente de gradient !

Voici quelques exemples d'hyperparamètres que l'on change en espérant améliorer les résultats de l'algorithme.

2.3.1 Taux d'apprentissage

Comme on l'a vu, le taux d'apprentissage η sert à contrôler le pas de descente de gradient. Or, en le gardant constant, on risque fort de manquer le minimum (local ou global) de la fonction de coût. S'il est trop faible, l'apprentissage est trop long, et la descente de gradient pourrait s'arrêter prématurément. En revanche, s'il est trop élevé, la descente de gradient pourrait diverger ! En voici deux illustrations

[SCHEMAS]

Pour remédier à cela, on dispose de plusieurs solutions. La plus facile conceptuellement consiste à diminuer le taux d'apprentissage au fur et à mesure des itérations. Dans ce cas, les premières itérations

provoqueront de grands changements dans les paramètres, mais les dernières ne feront que peaufiner ces changements.

Autrement, on envisage aussi l'a version "avec inertie", dont l'objectif est de sortir des minima locaux. Les détails importent peu ici, on peut se contenter d'imaginer un ballon roulant sur une pente et prenant de la vitesse.

2.3.2 Fonction d'activation

Comme on a pu s'en apercevoir, les fonctions d'activation sont nombreuses et vairées. D'expérience, les chercheurs savent quelle fonction d'activation employer selon la place d'un neurone dans le réseau ou selon le type de problème résolu. Ainsi, la plupart du temps, les neurones de la couche de sortie peuvent avoir une fonction d'activation différente selon que le réseau réalise une classification ou une régression.

Au début des neurones non linéaires, la fonction sigmoïde $sigma : t \mapsto \frac{1}{1+e^{-t}}$ fut beaucoup utilisée. Cependant, ses propriétés ne sont pas restées intéressantes indéfiniment. On a donc cherché des fonctions non saturantes et facilitant l'apprentissage, telles que $ReLU : t \mapsto max(0, t)$.

Assez vite, on a élaboré des variations d'après ReLU, comme "leaky ReLU" ou "parametric ReLU" ($PReLU : t \mapsto max(t, \alpha t)$), où α est, étrangement, un paramètre appris en même temps que tous les paramètres. Parfois, la frontière entre paramètre du modèle et paramètre de l'algorithme est brouillée. Il n'en demeure pas moins que la fonction d'activation est considérée comme un hyperparamètre.

2.3.3 Taille du réseau de neurones

La taille d'un réseau de neurones a un impact flagrant sur les performances de celui-ci, tant en terme de temps machine qu'en terme de précision dans ses prédictions. Un grand réseau peut à la fois être très lent à entraîner et sur-ajuster les données. D'une certaine manière, plus on a de neurones dans le réseau, plus on risque de mal apprendre, mais ne pas en avoir assez n'est pas idéal non plus ; le nombre de paramètres du modèle est donc un hyperparamètre de l'algorithme.

De manière assez remarquable, il existe un compromis entre mettre en place un gros réseau et un réseau plus modeste : il s'agit du ***décrochage*** (***dropout*** en anglais). Il s'agit simplement, pendant l'apprentissage, de virtuellement considérer comme absents certains neurones du réseau.

Plus généralement, cette technique fait partie des méthodes de ***régularisation***, dont le but est de pénaliser les modèles complexes au profit des plus simples. Cela peut se traduire par exemple en ajoutant une valeur maximale aux paramètres du modèle. Encore une fois, l'idée est de réduire du mieux possible le sur-ajustement

2.4 Conclusion

Les réseaux de neurones sont de formidables machines à optimiser des fonctions. Mieux, ce sont des machines à approcher des fonctions, et leur structure hautement non linéaire permettent souvent d'y arriver.

[FINIR]

On pressent que l'entrée d'un réseau de neurones devra être numérique pour que les algorithmes impliqués dans l'apprentissage fonctionnent. Ceci aura un impact important dans toute application de ces réseaux à des problèmes réel et divers.

Chapitre 3

Traitement du Langage Naturel

Après avoir décrit les principes de l'apprentissage par un réseau de neurones, voyons comment on peut les mettre en œuvre. On se restreint à des problèmes impliquant du texte, et plus précisément les problèmes de traitement automatisé du langage naturel. Explorons deux aspects complémentaires de l'apprentissage automatisé, que sont la classification et la génération de textes [1].

Pour les tâches d'apprentissage, on suppose que chaque auteur écrit dans un style qui lui est propre, reconnaissable par une machine. Le but secondaire de ce projet a été de vérifier cette hypothèse, sachant que le but principal était de découvrir l'apprentissage profond.

3.1 Les données

Nous avons souhaité découvrir les possibilités offertes par l'apprentissage profond pour des données textuelles. Pour ce faire, nous avons construit un corpus de plus de 150 romans de la littérature française classique, dont les auteurs sont Balzac, Dumas, Hugo et Zola (en plus des "Aventures de Télémaque" de Louis Aragon et de "L'Atelier de Marie-Caire de [QUI ? ?] Audoux). Bien évidemment, d'autres auteurs sont envisageables, mais ceux-ci ont l'avantage d'avoir été prolixes : plus de 80 romans pour Balzac, plus de 50 pour Zola ; Hugo et Dumas en ont écrit moins, mais très volumineux.

Nous avons utilisé des ressources libres pour récupérer les fichiers texte, à savoir gutenberg.org ainsi que wikimedia.org. Ces sites internet reproduisent de très nombreuses œuvres de manière numérique. Toutefois, il a fallu extraire chaque texte à la main, car, sur le premier site, toute navigation automatisée est interdite, alors que le second site n'a pas une structure suffisamment régulière d'un livre à l'autre pour le faire autrement. Dans une moindre mesure, nous avons aussi ôté toutes les annotations, remarques et certaines parenthèses des textes. Les explications de l'auteur ou de l'éditeur rompent le fil narratif de l'œuvre, en plus d'introduire des caractères spéciaux comme "[". Tous les textes ainsi copiés et pré-traités sont ensuite regroupés par auteur dans un répertoire "textes bruts".

En parcourant les romans écrits par chacun des auteurs cités, on remarque des cas atypiques. Le plus flagrant est "Han d'Islande", écrit par Victor Hugo. Ce roman regorge de mots et de phrases en islandais. Bien sûr, pour le lecteur, tout est traduit en note de bas de page ou entre parenthèses. Cependant, il serait mal venu d'utiliser ce (long) texte pour un algorithme cherchant à reconnaître un auteur. D'une part, enlever toutes les annotations serait à faire manuellement. D'autre part, le corpus se compose d'auteurs classiques français. Pour peu qu'on cherche à construire une représentation sémantique des mots du corpus, on trouverait tout à coup des termes étrangers synonymes de termes français ; typiquement une erreur de surinterprétation dans notre cas.

Les deux tâches d'apprentissage utilisent les textes différemment. Pour la génération de texte, il suffit de prendre un texte suffisamment long et le passer à l'algorithme. Nous en décrirons le processus en dernier. Pour ce qui est de la tâche de classification, présentée en premier, il a fallu découper les textes bruts. Leur taille initiale varie entre 20 Ko et 3 Mo, ce qui est très déséquilibré. On emploie

1. Ces problèmes se retrouvent le plus souvent mis en lumière avec le traitement d'images, mais de récentes expérimentations ont permis à des équipes de chercheurs d'imiter le style d'écriture d'une journaliste. Le succès fut tel que la technologie employée a été gardée secrète, car jugée trop dangereuse si employée à mauvais escient.

alors un script python pour découper les textes en extraits de 300 mots environ. Ce script se charge aussi de labelliser chacun des extraits avec le nom de son auteur. En voici un exemple :

" hugo

L'équipage était occupé à enverguer les voiles. Le gabier chargé de prendre l'empointure du grand hunier tribord perdit l'équilibre. On le vit chanceler, la multitude amassée sur le quai de l'Arsenal jeta un cri, la tête emporta le corps, l'homme tourna autour de la vergue, les mains étendues vers l'abîme ; il saisit, au passage, le faux marchepied [...] "

3.2 Classification de textes

3.3 Génération de texte

Conclusion