Université de Toulouse

Notes de cours.
Optimisation Stochastique et Apprentissage.

Pierre Weiss

Table des matières

Avant-propos

L'optimisation stochastique est un domaine assez ancien. Un des premiers papiers proposant d'utiliser le gradient stochastique date de 1951 avec l'algorithme de Robbins-Monro [1]. Cependant, ce domaine est resté relativement confidentiel pendant une bonne soixantaine d'année (bien qu'une réelle communauté talentueuse et visionnaire ait existé continuellement). La raison principale est probablement la puissance modérée des ordinateurs qui rendaient ce genre de technique assez peu utiles. L'évolution des mémoires, des puissances de calcul, de la parallélisation, ainsi que les nouvelles applications en apprentissage machine a rendu ces techniques beaucoup plus attrayantes ces 20 dernières années. C'est donc un domaine en pleine explosion depuis 2010 environ, qui va probablement voir de beaux progrès dans les années à venir. Contrairement à beaucoup de cours, celui-ci contiendra donc des références très récentes et il est possible que certaines idées proposées ici soient dépassées dans les mois ou années à venir. De plus, ma faible connaissance du domaine va forcément donner une vision un peu étriquée. J'encourage vivement les étudiants motivés à lire certains livres ou articles (certains sont proposés en bibliographie) pour se forger une vision plus riche.

Ces outils sont cependant devenus centraux en ingénierie avec de très nombreuses opportunités d'applications nouvelles et de postes intéressants. J'ai donc fait mon possible pour présenter des outils qui me semblent utiles, même s'ils sortent un peu des sentiers battus. Couplé au fait que je ne sois pas un expert de ce domaine naissant, mais plutôt un amateur amusé, je ne saurais trop vous conseiller d'être très attentifs aux progrès de ce domaine si vous y travaillez.

Ce cours a été construit à partir de nombreuses références et cours en ligne. Les références principales sont [2, 3, 4, 5, 6]. J'ai aussi utilisé plusieurs notes de cours en ligne, dont celles de Francis Bach et de Clément Royer.

Le format livre de ce document est probablement inadapté à ces petites notes écrites en quelques jours. Il est cependant possible qu'elles enflent au cours du temps si je suis amené à perpétuer mes enseignements dans ce domaine.

Chapitre 1

Introduction

Dans ce chapitre, nous posons le cadre général de ce cours. Je me concentre sur le domaine de l'apprentissage supervisé, car c'est aujourd'hui celui qui dynamise le plus ce domaine. Cependant, il y a des applications dans de nombreux autres domaines. Je ne les détaillerai pas ici, principalement par manque de temps.

1.1 Apprentissage machine supervisé

Commençons par définir de façon abstraite ce qu'est un algorithme d'apprentissage supervisé. Soit $\mathcal{X}$ un espace d'entrées et $\mathcal{Y}$ un espace de sorties. On note $\mathcal{Z}$ l'espace des couples $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. Supposons maintenant qu'on dispose d'une loi de probabilité $P_{X,Y}$ définie sur $\mathcal{Z}$. Ceci permet de définir un couple aléatoire $Z = (X, Y)$ qui suit la loi $P_{X,Y}$. La distribution conditionnelle $P_{Y|X}$ représente la relation inconnue entre les entrées et les sorties. De façon générale, l'objectif de l'apprentissage supervisé est d'apprendre cette loi conditionnelle, qui permet de prédire Y connaissant X. Pour construire un prédicteur $h : \mathcal{X} \to \mathcal{Y}$ qui prédise Y à partir X, il est préférable que les variables aléatoires X et Y aient des dépendances fortes du type $Y \approx T(X)$ où T est une application déterministe qu'on souhaite évaluer. Les quelques exemples suivants sont triés par ordre décroissant de déterminisme entre les variables X et Y :

1. Déterminer le résultat d'une équation (e.g. $\frac{\partial y}{dt} = \text{div}(x \nabla y)$), à partir du coefficient de diffusion x. On peut effectivement apprendre à calculer la solution de certaines équations aux dérivées partielles au lieu d'utiliser des schémas numériques (attention, dans beaucoup de cas, on n'a pour le moment pas le choix et il faut avoir recours à de l'analyse numérique fine).
2. Déterminer un label "tumeur benigne" ou "tumeur maligne" à partir d'une image de tissu. La plupart du temps, la réponse est claire, c'est à dire, que l'application T est bien définie, c'est à dire qu'il y a peu de doutes sur l'état du tissu. Cependant, dans d'autres, la réponse est incertaine (i.e. $P_{Y|X}$ n'est pas une masse de Dirac).
3. Déterminer la conformation (structure 3D) d'une protéine Y, à partir de la séquence d'acide aminés X. C'est le principe d'Alphafold, prix Nobel 2024.

4. Déterminer la météo Y du lendemain, à partir de mesures en stations météorologiques X.
5. Synthétiser une image Y à partir d'un prompt X (e.g. dessine moi un cavalier sur la lune, voire ChatGPT ou Dall-E). Dans ce cas, il existe énormément d'images qui coïncident avec la requête. Ainsi, il est important de réellement approcher la loi jointe $P_{Y|X}$.

Dans ce cours, on va principalement s'intéresser aux cas déterministes, où construire une *fonction de prédiction* h a un sens. Pour construire la loi conditionnelle $P_{Y|X}$ complète, il faut avoir recours à des techniques dites de diffusion, qui ne rentrent pas dans le cadre de ce cours.

Une manière d'évaluer la qualité d'un prédicteur h est d'évaluer le risque empirique. Etant donnée une fonction de perte (fonction loss)

$$\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R} \cup \{+\infty\},$$

qui mesure la différence entre la prédiction $h(X)$ et la vraie sortie Y, on peut quantifier la précision d'une fonction h par le *risque moyen* :

$$E(h) \overset{\text{def.}}{=} \int \ell(h(x), y) dP(x, y) = \mathbb{E}(\ell(h(X), Y)). \tag{1.1}$$

Par la suite, on notera h^* la fonction qui minimise E sur toutes les applications de $\mathcal{X}$ dans $\mathcal{Y}$, i.e.

$$h^* = \underset{h:\mathcal{X}\to\mathcal{Y}}{\arg\min}\, E(h).$$

La fonction h^* peut ne pas exister (un minimum n'est pas toujours atteint), mais surtout, elle ne peut pas être évaluée numériquement pour deux raisons :

1. la loi P est inconnue et l'espérance (1.1) ne peut donc pas être évaluée numériquement.
2. l'espace de toutes les applications de $\mathcal{X}$ dans $\mathcal{Y}$ est infini à moins que $\mathcal{X}$ et $\mathcal{Y}$ soient eux-mêmes finis.

L'apprentissage machine repose donc sur deux ingrédients importants :

Risque empirique On suppose qu'on dispose d'un nombre n fini de données d'apprentissage $(x_n, y_n)_{1\leq n\leq N}$ qui ont été tirées indépendamment suivant la loi P. On peut alors remplacer alors le risque moyen (1.1) par le *risque empirique* E_n défini par :

$$E_N(h) \overset{\text{def.}}{=} \frac{1}{N} \sum_{n=1}^{N} \ell(h(x_n), y_n). \tag{1.2}$$

Régularisation Pour se ramener à un problème en dimension finie, on va minimiser E_n sur une famille $\mathcal{H}$ d'applications encodées par un nombre $D \in \mathbb{N}$ fini de paramètres. Cette famille peut être par exemple un espace de polynômes, un réseau de neurones,... On est donc ramenés à la minimisation d'un problème de la forme :

$$\min_{h\in\mathcal{H}} E_N(h) = \frac{1}{N} \sum_{n=1}^{N} \ell(h(x_n), y_n). \tag{1.3}$$

On peut aussi considérer des régularisations par pénalisation de critère : on souhaite minimiser une fonction de régularisation $R(h)$ qui va favoriser certaines propriétés de la solution. On est alors amenés à des problèmes du type :

$$\min_{h\in\mathcal{H}} E_N(h) = \frac{1}{N}\sum_{n=1}^{N} \ell(h(x_n), y_n) + R(h). \tag{1.4}$$

Dans ce cours, nous nous poserons notamment les questions suivantes :
— Quel est l'effet de minimiser le risque empirique E_N plutôt que le risque moyen E ?
— Quel est l'effet de la régularisation par la famille $\mathcal{H}$ ou de la pénalisation $R(h)$?
— Comment minimiser le risque (1.3) et ce malgré les dimensions potentiellement gigantesques N et D ? Est-ce possible ?
— Quelles garanties théoriques peut-on donner lorsque le risque empirique E_n est convexe ? Quand il est non convexe ?
— Quelle est la précision de minimisation nécessaire ?

Ces questions sont cruciales dans la majorité des tâches d'apprentissage machine, et les "data-scientists" ou utilisateurs de librairies telles que TensorFlow, Keras ou PyTorch y sont confrontés au quotidien.

1.2 Quelques exemples

Avant d'aller plus loin, nous allons essayer de rendre les concepts ci-dessus un peu moins abstraits en montrant quelques applications.

1.2.1 Reconnaissance de caractères (classification)

Une base de données célèbre - pour la conception de TPs notamment - est la base de données MNIST. Celle-ci contient des images de chiffres de 0 à 9, voir Fig. 1.1. Les centres postaux doivent trier des milliards de lettres chaque jour dans le monde et des outils de reconnaissance automatique permettraient de rediriger le courier efficacement. Cependant, comment construire un algorithme de reconnaissance de caractères ?

Le choix des espaces $\mathcal{X}$ et $\mathcal{Y}$ et de la fonction perte pour cette application peuvent être les suivants :
— N est le nombre d'exemples, ici 60000 caractères.
— $\mathcal{X}$ représente l'ensemble des imagettes, i.e. $\mathcal{X} = \mathbb{R}^{28\times 28}$.
— $\mathcal{Y}$ est l'espace des nombres possibles, i.e. $\mathcal{Y} = \{0, 1, \dots, 9\}$.
— Un choix naturel pour la fonction perte ℓ est $\ell(\hat{y}, y) = \begin{cases} 0 & \text{si } y = \hat{y} \\ 1 & \text{sinon} \end{cases}$. Cependant, il n'est pas bon pour l'optimisation différentiable : cette fonction ℓ est constante presque partout et discontinue. Ainsi, elle ne se prête pas du tout à l'optimisation par descente de gradient.

Ainsi, la solution dans ce cas, est de chercher la loi conditionnelle $P_{Y|X}$ qui est un vecteur de probabilité, ce qui nous mène à introduire le simplexe.

FIGURE 1.1 – Quelques éléments de la base de données MNIST . Chaque caractère est une imagette 28×28.

Définition 1.1 (Simplexe) *Le simplexe de $\mathbb{R}^{M-1}$ représente l'espace des probabilités discrètes sur $\mathbb{R}^M$. Il est défini comme :*

$$\Delta_{M-1} = \{p \in \mathbb{R}^M, p_m \geq 0 \ \forall m, \sum_{m=1}^{M} p_m\}. \tag{1.5}$$

Ainsi, la fonction de prédiction h est définir par $h : \mathcal{X} \to \Delta_9$ et $(h(x))_m$ représente la probabilité que l'image x représente le nombre m. Ce choix donne lieu aux espaces suivants :

- $\mathcal{X}$ représente l'ensemble des imagettes, i.e. $\mathcal{X} = \mathbb{R}^{28\times 28}$.
- $\mathcal{Y}$ est l'espace des probabilités sur $\{0, \ldots, 9\}$, i.e. $\mathcal{Y} = \Delta_9$.
- y_n est un vecteur de $\mathbb{R}^{10}$ qui vaut 1 à la coordonnée correspondant au chiffre que représente x_n et 0 ailleurs.
- Plusieurs fonctions de perte peuvent alors être utilisées, qui représentent des distances ou divergence entre vecteurs de probabilité discrète (e.g. entropie croisée, Kullback-Leibler, variation totale, ...). Nous y reviendrons dans quelques lignes.

Le choix de la famille de fonctions $\mathcal{H}$ est important et nous verrons quelques possibilités un peu plus loin.

1.2.2 Résolution de problèmes inverses (régression)

Dans un problème inverse, on souhaite reconstruire un signal x dans une famille $\mathcal{X}$. Cette famille peut représenter des images (e.g. échographe, imageur par résonance magnétique, microscope) ou encore des familles de champs de vecteurs sur la sphère (e.g. prédiction des vents en météorologie). Ces signaux sont mesurés indirectement à partir d'un opérateur linéaire $L : \mathcal{X} \to \mathcal{Y}$ où $\mathcal{Y} = \mathbb{R}^M$ représente un ensemble de mesures de x (e.g. coefficients de Fourier du signal en quelques points, mesures de pression ou de températures). Ainsi, on est confrontés au problème d'estimation de x_0 à partir de $y = Lx_0$. Il faut inverser l'opérateur L, ce qui est malheureusement impossible car il possède un

noyau. La façon usuelle de reconstruire les données consiste alors à résoudre des problèmes d'optimisation du type :

$$\inf_{x\in\mathcal{X}, Lx=y} R(x), \tag{1.6}$$

où $y \in \mathbb{R}^M$ est un ensemble de mesures et R est un terme de régularisation qui permet de sélectionner un élément dans l'ensemble $L^{-1}y$. Le choix de ce régulariseur est très difficile en pratique et a mené à des théories élégantes et efficaces (e.g. échantillonnage compressé et régularisation ℓ^1).

Une tendance récente est d'apprendre directement un reconstructeur plutôt que de résoudre le problème d'optimisation (1.6). Pour ce faire, on se donne une collection $(x_1, \dots, x_N)$ d'images et on leur associe leurs mesures $(y_1, \dots, y_N)$ avec $y_n = Lx_n$. On construit ensuite une famille de reconstructeurs $\mathcal{H}$. Chaque reconstructeur h est une fonction de $\mathcal{Y} = \mathbb{R}^M$ dans $\mathcal{X}$. On peut entraîner un reconstructeur en minimisant :

$$\inf_{h\in\mathcal{H}} \frac{1}{N}\sum_{n=1}^{N} \ell(h(y_n), x_n).$$

Dans ce cas, la fonction perte ℓ est typiquement définie par $\ell(\hat{x}, x) = \|\hat{x} - x\|$ où $\|\cdot\|$ est une norme sur l'espace $\mathcal{X}$.

1.3 Quelques exemples de fonctions perte

Le choix de la fonction perte $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R} \cup \{+\infty\}$ dépend de l'application. Le choix va différer suivant qu'on est face à des tâches de classification ou de régression.

1.3.1 Classification

La classification permet de prédire des *labels*, ceci signifie que l'espace $\mathcal{Y}$ est discret. Par exemple, en médecine on peut s'intéresser à des questions binaires du type : ce patient est-il malade ou pas ? Dans ce cas, on peut poser $\mathcal{Y} = \{\text{malade}, \text{pas malade}\}$, ou plus simplement $\mathcal{Y} = \{-1, 1\}$. En vision par ordinateur, on peut se demander : quel est l'animal contenu dans cet enclos ? Dans ce cas, on peut poser $\mathcal{Y} = \{\text{Léopard, Baleine, Ours, Lézard, Humain}\}$, ou plus simplement $\mathcal{Y} = \{1, 2, 3, 4, 5\}$.

Dans ces conditions, la fonction perte la plus naturelle est

$$\ell(\hat{y}, y) = \begin{cases} 0 & \text{si } \hat{y} = y \\ 1 & \text{sinon.} \end{cases} \tag{1.7}$$

Malheureusement, cette fonction est non continue, non différentiable, constante presque partout. Bref, elle n'est pas minimisable par des méthodes d'optimisation continue et il faut avoir recours à des techniques combinatoires qui ne sont pas l'objet de ce cours. En pratique, on lui préfère donc souvent des approximations différentiables et si possible convexes. Dans ce cas, y peut vivre dans un espace discret, alors que $\hat{y}$ vit dans un espace continu. Typiquement, pour de la classification binaire, correspondant à l'ensemble $\mathcal{Y} =$

$\{-1, 1\}$, on va construire un prédicteur $h : \mathcal{Y} \to \mathbb{R}$ et définir le label de $h(y)$ par son signe. Des choix populaires et convexes pour la classification binaire sont :

Fonction $0-1$ **:** $\ell(\hat{y}, y) = \begin{cases} 0 & \text{si signe}(\hat{y}) = \text{signe}(y) \\ 1 & \text{sinon} \end{cases}$. Cette fonction ne doit pas être utilisée avec des méthodes d'optimisation continue car son gradient est nul presque partout.

Fonction "hinge" : (fonction charnière) $\ell(\hat{y}, y) = \max(0, 1 - \hat{y}y)$. Elle est notamment utilisée dans les Support Vector Machines (SVM).

Je renvoie le lecteur intéressé à `https://en.wikipedia.org/wiki/Loss_functions_for_classification` pour plus de détails et un bestiaire plus riche.

En classification multi-labels, des fonctions pertes populaires sont l'entropie croisée binaire ou l'entropie croisée sigmoïde. Supposons qu'on doive choisir un label parmi m (e.g. $\mathcal{Y} = \{1, 2, \ldots, m\}$). La méthode dominante pour réaliser cette tâche est de définir un prédicteur vectoriel $h : \mathcal{X} \to \Delta^{M-1}$. Pour qu'une application renvoie des valeurs sur le simplexe, on peut d'abord construire une application sur $\mathbb{R}^m$, puis la transférer sur le simplexe en utilisant l'application :

$$[f(z)]_j = \frac{\exp(z_m)}{\sum_{m=1}^{M} \exp(z_m)}. \tag{1.8}$$

Cette fonctions a plusieurs avantages :

— Elle rend tous les coefficients positifs et leur somme vaut 1. Elle envoie donc dans l'espace des probabilités discrètes et on peut interpréter les coefficients de prédicition comme une probabilité d'appartenance à une classe.

— Elle "écarte" les valeurs de z. En particulier, si un coefficient de z est plus grand que les autres, il devient significativement plus grand par passage dans l'exponentielle. Ainsi, l'application f favorise les vecteur parcimonieux, où un seul coefficient est grand. C'est une propriété souhaitable : on aimerait qu'un prédicteur favorise un label plutôt qu'un autre.

Une fois les valeurs sur le simplexe, on peut utiliser n'importe quelle distance (ou divergence) entre probabilités. Supposons que le vrai label de la donnée x_n soit donné par un vecteur $y_n \in \{0, 1\}^M$ avec

$$y_n[m] = \begin{cases} 1 & \text{si label}(x_n) = m \\ 0 & \text{sinon.} \end{cases} \tag{1.9}$$

Supposons aussi que le prédicteur renvoie un vecteur $\hat{y}_n \in \Delta^{M-1}$. La fonction la plus populaire en classification pour comparer y_n à $\hat{y}_n$ est l'entropie croisée (cross-entropy) définie par

$$\ell(\hat{y}_n, y_n) = -\sum_{m=1}^{M} y_n[m] \log(\hat{y}_n[m]).$$

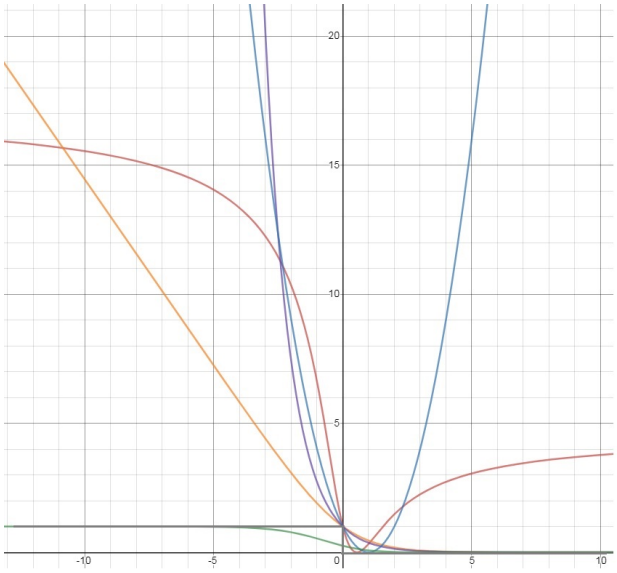

FIGURE 1.2 – Exemples de fonction perte pour la classification binaire . Gris : 0-1, Vert : savage, Orange : logistique, Violet : exponentielle, Brun : tangente, Bleu : quadratique.

1.3.2 Régression

En régression, le problème est de prédire des quantités continues, comme des valeurs de $\mathbb{R}^m$. C'est par exemple le cas de la résolution de problèmes inverses. Dans ce cas, les fonctions pertes typiques sont du type :

$$\begin{aligned} \textit{Quadratique :} \quad & \ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_2^2. \\ \textit{Robuste :} \quad & \ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|_1. \\ \textit{Logistique :} \quad & \ell(\hat{y}, y) = \log(1 + \exp(-(\hat{y} - y))). \\ \textit{Réseaux neuronaux :} \quad & \ell(\hat{y}, y) = NN(\hat{y}, y). \end{aligned}$$

Dans la dernière équation, $NN : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ est un réseau de neurone entraîné pour quantifier la qualité d'une prédiction. Par exemple, en imagerie, on peut construire des réseaux de neurones capables de définir une qualité d'images similaire à ce que serait l'évaluation d'un humain. La figure 1.2 montre quelques-unes de ces fonctions perte.

A travers ces quelques exemples populaires, on peut voir que les propriétés des fonctions perte varient beaucoup : être convexe ou non convexe, différentiables ou non différentiables. Il faudra donc construire des méthodes d'optimisation et des théories capables de s'adapter à ces difficultés.

1.4 Quelques exemples de familles de fonctions

Jusqu'à présent, nous avons passé sous silence le choix de la famille $\mathcal{H}$. Nous décrivons quelques choix populaires ci-dessous. De façon générale, une famille $\mathcal{H}$ est définie par un

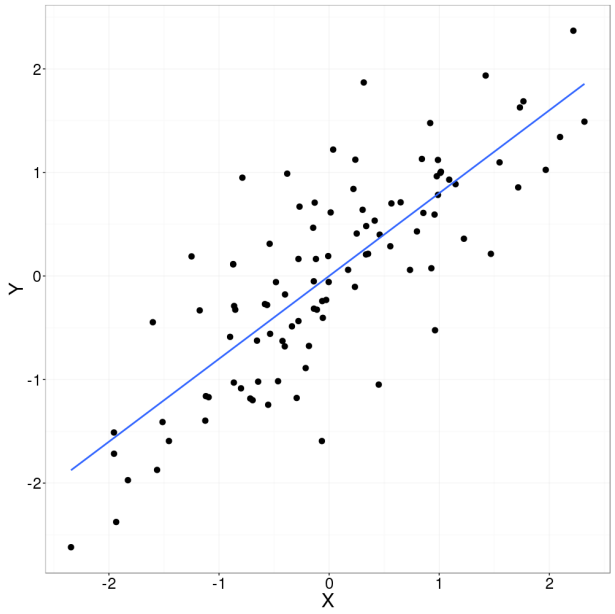

FIGURE 1.3 – La régression linéaire en dimension 1.

ensemble de fonctions paramétrées

$$\begin{aligned} h :& \mathcal{X} \times \mathcal{W} \to \mathcal{Y} \\ & (x, w) \mapsto h(x, w) \end{aligned}$$

où le vecteur $w \in \mathbb{R}^D$ représente les poids (weights) de la fonction. Entraîner le prédicteur, c'est donc optimiser les poids $w \in \mathbb{R}^D$.

1.4.1 Régression et classification linéaire

Régression En régression linéaire ou affine, on va poser $\mathcal{X} = \mathbb{R}^P$ et $\mathcal{Y} = \mathbb{R}^Q$. La régression linéaire permet alors d'évaluer des lois où les variables X et Y sont liées par une relation du type $Y \simeq AX + b$ où $A \in \mathbb{R}^{P \times Q}$ est une matrice et $b \in \mathbb{R}^Q$ un vecteur. Dans ce cadre, les poids w définis précédemment correspondraient au couple (A, b) et la dimension $D = P \cdot Q + Q$. Un exemple en dimension 1 est donné sur la figure 1.3. Ce cadre est probablement le mieux compris des points de vue statistiques et d'optimisation. Il est particulièrement intéressant car il est simple et permet de comprendre de nombreuses facettes de l'apprentissage. En grande dimension, il permet déjà de faire apparaître des phénomènes non triviaux.

Classification binaire Les prédicteurs linéaires peuvent aussi être utilisés pour effectuer de la classification. On se focalise sur le cas de la classification binaire où $\mathcal{X} = \mathbb{R}^P$ et $\mathcal{Y} = \{-1, 1\}$. Dans ce cadre, on peut définir un prédicteur par

$$h(y) = \text{signe}(w^T x) \tag{1.10}$$

où $w \in \mathbb{R}^D$ avec $d = p$ représente des poids à optimiser. Dans beaucoup d'applications, on préfère transformer les données d'entrées x par une application (non)-linéaire $\Phi : \mathbb{R}^P \to \mathbb{R}^D$ et le prédicteur devient alors

$$h(y) = \text{signe}(w^T \Phi(x)). \tag{1.11}$$

L'application Φ est en général choisie avec $D \gg P$. L'idée sous-jacente est qu'il est bien plus facile de séparer des données en grande dimension qu'en petite dimension.

1.4.2 Réseaux neuronaux

Un choix qu'on ne peut pas passer sous silence tellement il est devenu populaire et performant est celui des réseaux de neurones. Ceux-ci peuvent être vus comme une série de transformations linéaires et non linéaires paramétrées. Etant donnée une entrée x, le réseaux de neurone génère une suite

$$\begin{aligned} x = x^{(0)} &\overset{\mathcal{A}_0}{\mapsto} x^{(1)} \overset{\mathcal{A}_1}{\mapsto} \dots \overset{\mathcal{A}_{J-1}}{\mapsto} x^{(J)} \\ &= \mathcal{A}_{J-1}\left(\dots \mathcal{A}_1(\mathcal{A}_0(x^{(0)}))\right) \end{aligned}$$

où J représente la profondeur du réseau et les applications $\mathcal{A}_j$ sont des applications linéaires ou non linéaires qui représentent les différentes couches du réseau. Des choix les plus populaires pour la structure de ces applications sont :

Réseaux linéaires : $\mathcal{A}_j(x_{j-1}) = A_j x_{j-1} + b_j$ où $A_j \in \mathbb{R}^{P_j \times P_{j-1}}$ est une matrice et $b_j \in \mathbb{R}^{p_j}$ un vecteur appelé biais. De tels réseaux permettent de reproduire des techniques de factorisation matricielle qui sont essentiels pour le calcul de transformées telles que la FFT (Fast Fourier Transform) ou les transformées en ondelettes.

Réseaux non-linéaires : $\mathcal{A}_j(x_{j-1}) = \sigma(A_j x_{j-1} + b_j)$ où $A_j \in \mathbb{R}^{P_j \times P_{j-1}}$ est une matrice et $b_j \in \mathbb{R}^{P_j}$ un vecteur et où $\sigma : \mathbb{R}^{P_j} \to \mathbb{R}^{P_j}$ est une fonction d'activitation telle que la fonction RELU ou une sigmoïde.

Réseaux convolutionels : $\mathcal{A}_j(x_{j-1}) = \sigma(\psi_j \star x_{j-1} + b_j)$ où $\psi_j \in \mathbb{R}^P$ est un filtre de convolution $b_j \in \mathbb{R}^P$ un vecteur et où $\sigma : \mathbb{R}^P \to \mathbb{R}^P$ est une fonction d'activitation.

Dans ce cas, les poids w à entraîner sont les paramètres du réseau, c'est-à-dire la séries de matrices et biais $(A_j, b_j)_{0 \le j \le J}$ ou la série de filtres et de biais $(\psi_j, b_j)_{0 \le j \le J}$ pour les réseaux convolutionnels.

1.5 Régularisation

Finalement, il est souvent essentiel de régulariser les poids w pour éviter le phénomènes de sur-apprentissage (overfitting), notamment si le nombre d de paramètres à apprendre est très grand. En pratique, on ajoute donc souvent des termes de régularisation $R(w)$ du type suivant :

Quadratique : $R(w) = \frac{1}{2}\|w\|_2^2$. Ces termes ont beaucoup de qualités :
— Ils sont faciles à traiter numériquement (forte convexité + différentiabilité).

— La théorie associée est très riche (théorie des noyaux reproduisants, théorèmes des représentants).

Normes ℓ^1 : le choix le plus emblématique est $R(w) = \|w\|_1$. Là encore, ce terme a de nombreux avantages :
— Il est convexe, non différentiable.
— Il favorise la parcimonie des solutions.
— Une théorie riche a été développée depuis 2004. On dispose maintenant de théorèmes de représentants.
— Ce type de terme favorise la parcimonie et permet de faire ce qu'on appelle de la sélection de modèle.

En termes d'optimisation, l'ajout d'un régulariseur peut rendre la résolution du problème plus simple (e.g. en assurant la forte convexité de l'énergie) ou plus difficile (e.g. en ajoutant des non différentiabilités).

1.6 Quelques exemples d'énergies

Pour conclure, nous donnons ci-dessous quelques exemples de fonctions typiques que nous pourrions vouloir optimiser. Les propriétés de ces fonctions (régularité et convexité) rendent la minimisation a priori de plus en plus complexe.

Régression linéaire Etant donnée une série de données d'entrées $x_n \in \mathbb{R}^P$ et de sorties $y_n \in \mathbb{R}^Q$, on va chercher les poids $w = (A, b)$ qui minimisent :

$$\inf_{A \in \mathbb{R}^{Q \times P}, b \in \mathbb{R}^Q} \frac{1}{2} \sum_{n=1}^{N} \|Ax_n + b - y_n\|_2^2.$$

Cette énergie est convexe, différentiable. Les conditions d'optimalité montrent que les solutions sont obtenues en résolvant un système linéaire. Pour rendre la solution unique et régulariser le problème, on peut considérer une régularisation ℓ^2 :

$$\inf_{A \in \mathbb{R}^{Q \times P}, b \in \mathbb{R}^Q} \frac{1}{2} \sum_{n=1}^{N} \|Ax_n + b - y_n\|_2^2 + \lambda \left(\|A\|_F^2 + \|b\|_2^2 \right),$$

où λ est un paramètre de régularisation et où $\|A\|_F$ est la norme de Frobenius de la matrice A. L'ajout du régulariseur assure l'unicité de la solution et rend la fonction fortement convexe.

Classification binaire linéaire et SVM On se place ici dans le cadre où $\mathcal{Y} = \{-1, 1\}$ et $\mathcal{X} = \mathbb{R}^P$. Etant donnée une fonction $\Phi : \mathbb{R}^P \to \mathbb{R}^D$ qui sert à mieux séparer les données (le "kernel trick"), on cherche un hyperplan H de la forme

$$H = \{x \in \mathbb{R}^p, w^T \Phi(x_n) - b = 0\}$$

qui sépare "au mieux" les données (x_n) en classes 1 et -1. Pour ce faire, on peut résoudre :

$$\inf_{w\in\mathbb{R}^D} \frac{1}{2}\|w\|_2^2 \quad \text{tel que} \quad \begin{cases} w^T\Phi(x_n) - b \geq 1 & \text{si } y_n = 1 \\ w^T\Phi(x_n) - b \leq -1 & \text{si } y_n = -1 \end{cases}.$$

Ce problème correspond à une des formulations des Support Vector Machine (SVM). Je renvoie le lecteur intéressé à la page Wikipédia ou aux autres cours d'apprentissage pour des informations complètes sur cette formulation. Le problème ci-dessus est convexe. Il consiste à minimiser une fonction quadratique sur un polytope convexe (c'est de la programmation quadratique).

Si les données ne sont pas séparables, l'ensemble des contraintes peut être vide et le problème ci-dessus n'admet alors pas de solution. Dans ce cas, on peut remplacer les contraintes par une fonction du type :

$$\inf_{w\in\mathbb{R}^D} \frac{\lambda}{2}\|w\|_2^2 + \sum_{n=1}^{N} \max(0, 1 - y_n w^T\Phi(x_n)),$$

où $\lambda > 0$ est un paramètre qui permet d'équilibrer les deux fonctions. Là encore, cette énergie est convexe, mais non différentiable à cause de la fonction max.

Réseaux de neurones Pour entraîner un réseau de neurones, on commence par fixer sa structure. Par exemple, dans les exemples de la section 1.4.2, on pourrait fixer la profondeur du réseau J ainsi que la liste des dimensions (P_j). On obtient ainsi un réseau paramétré $h(x, w)$ où les poids w correspondent aux coefficients des applications linéaires $\mathcal{A}_j$. La dimension d des poids du réseau peut devenir gigantesque (e.g. plusieurs dizaines de millions) de paramètres. On est alors menés à minimiser des énergies de type :

$$\inf_{w\in\mathbb{R}^D} \sum_{n=1}^{N} \ell(h(x_n, w), y_n) + R(w),$$

où R est une fonction de régularisation des poids. Ce type d'énergie est le cauchemar (ou le gagne-pain) des optimiseurs. Les énergies sont nonconvexes, non différentiables (si on utilise des fonctions RELU) en très grande dimension.

Chapitre 2

Les erreurs d'apprentissage

Ce chapitre repose en grande partie sur le papier [3]. Il a pour objectif de montrer les spécificités en termes de précision des problèmes d'optimisation en apprentissage.

2.1 L'équilibre des erreurs

Comme on l'a vu en introduction, un objectif "idéal" serait de calculer le minimiseur

$$h^* \in \operatorname*{arg\,min}_{h:\mathcal{X}\to\mathcal{Y}} E(h)$$

du risque moyen (1.1). En pratique, on a cependant besoin de restreindre l'espace des prédicteurs à une famille $\mathcal{H}$ (e.g. régression linéaire, réseau de neurone) qui ne contient pas forcément le prédicteur idéal h^*. Ceci donne lieu à un minimiseur :

$$h^*_{\mathcal{H}} = \operatorname*{arg\,min}_{h\in\mathcal{H}} E(h)$$

potentiellement différent de h^*. Une deuxième source d'erreur est liée au fait qu'on minimise le risque empirique E_N plutôt que E. Ceci donne lieu à un prédicteur

$$h^*_N = \operatorname*{arg\,min}_{h\in\mathcal{H}} E_N(h)$$

différent de $h^*_{\mathcal{H}}$. Enfin, on ne calcule pas exactement le minimiseur h_N, car on utilise des algorithmes imparfaits. Ceci mène à une erreur $\mathcal{E}$ entre le prédicteur idéal h^* et le prédicteur calculé $\tilde{h}_N$ qui peut être définie par

$$\begin{aligned}\mathcal{E} &\overset{\text{def.}}{=} E(\tilde{h}_N) - E(h^*) \\ &= \underbrace{E(\tilde{h}_N) - E(h^*_N)}_{\mathcal{E}_{opt}} + \underbrace{E(h^*_N) - E(h^*_{\mathcal{H}})}_{\mathcal{E}_{est}} + \underbrace{E(h^*_{\mathcal{H}}) - E(h^*)}_{\mathcal{E}_{app}}.\end{aligned}$$

Les différentes erreurs sont décrites ci-dessous.

$\mathcal{E}_{app}$ l'erreur d'approximation $\mathcal{E}_{app}$ correspond au fait que la famille $\mathcal{H}$ de prédicteurs soit inadaptée à la tâche d'apprentissage. Cette erreur peut être diminuée en augmentant la taille de la famille $\mathcal{H}$ (i.e. en augmentant d). Cependant le prix à payer est une plus grande complexité d'optimisation. Contrôler l'erreur $\mathcal{E}_{app}$ est un domaine actif de recherche qui tient à des thèmes tels que la théorie de l'approximation, la géométrie des espaces, l'analyse harmonique ou la théorie de l'information. Nous évoquerons assez peu ce thème, même s'il est passionnant avec des progrès attendus à venir.

$\mathcal{E}_{est}$ l'erreur d'estimation $\mathcal{E}_{est}$ est d'autant plus petite que le nombre d'observations n est grand. Contrôler cette erreur est le domaine des statistiques. Il est typiquement attaqué avec des outils de concentration de la mesure. On les évoquera rapidement ci-dessous.

$\mathcal{E}_{opt}$ L'erreur d'optimisation $\mathcal{E}_{opt}$ correspond à l'erreur liée au fait qu'on utilise des algorithmes imparfaits. La plus grosse partie de ce cours sera dédiée à cet aspect. Par la suite, on supposera qu'on peut contrôler cette erreur en utilisant des algorithmes plus ou moins coûteux et on posera

$$\mathcal{E}_{opt} \leq \rho.$$

En pratique il ne sert à rien d'avoir une erreur d'optimisation nulle si l'erreur d'approximation ou si l'erreur d'estimation est très grande : il est suffisant que les erreurs soient du même ordre de grandeur. Ainsi, si le nombre d'observations n est petit, on peut imaginer que l'erreur d'estimation est grande et on pourra se contenter d'une méthode d'optimisation très approximative. C'est en ce sens, que l'optimisation (stochastique) en apprentissage se démarque de l'optimisation déterministe usuelle. Le tableau 2.1 résume les variations de précision en fonction des paramètres. Dans la suite de ce chapitre, nous brosserons une esquisse rapide du contrôle des erreurs d'approximation et d'estimation. Cette description sera très lacunaire : le contrôle de chaque terme est passionant et pourrait ainsi faire l'objet d'un cours entier.

TABLE 2.1 – Les variations des erreurs et du temps de calcul en fonction des tailles des paramètres.

	D (dim. modèle)	N (# échantillons)	ρ (prec. optim.)
$\mathcal{E}_{est}$ (erreur estimation)	...	$\searrow$	...
$\mathcal{E}_{app}$ (erreur approximation)	$\searrow$	...	...
$\mathcal{E}_{opt}$ (erreur d'optimisation)	...	...	$\nearrow$
T (temps de calcul)	$\nearrow$	$\nearrow$	$\searrow$

A retenir

Lorsqu'on effectue des tâches d'apprentissage, il est nécessaire de travailler simultanément sur la classe de modèles de prédicteurs (dimension D), sur la taille de la base de donnée (dimension N), puis d'utiliser un algorithme d'optimisation dont la précision dépend des deux paramètres précédents. *On peut souvent se contenter de solveurs peu précis.*

2.2 Erreurs d'approximation

Dans cette partie, nous expliquons quels outils mathématiques permettent de contrôler l'erreur d'approximation. Il est important de comprendre que si la fonction de prédiction idéale h^* était une application arbitraire de $\mathcal{X}$ dans $\mathcal{Y}$, il n'y aurait aucun espoir de l'estimer. Par exemple, considérons une fonction $h^* : [0,1] \to \mathbb{R}$ et supposons qu'on l'ait échantillonnée en certains points $x_i \in [0,1]$, ce qui nous a renvoyé les valeurs $y_i = h(x_i)$. A quelle condition est-il possible de reconstruire h^* à partir de cette information ? La théorie de l'échantillonnage fournit des réponses intéressantes. Par exemple, plus la fonction est régulière, plus une interpolation polynomiale va être précise. Au contraire, si la fonction est complètement erratique et présente des discontinuités imprévisibles d'un point à son voisin, il sera très difficile de l'interpoler correctement. Nous présentons quelques outils qui permettent de quantifier ces phénomènes ci-dessous.

2.2.1 L'importance de l'analyse

Dans le cas le plus simple, l'erreur d'approximation $\mathcal{E}_{app}$ est nulle. Il "suffit" pour obtenir une erreur nulle que la vraie fonction optimale h^* appartienne à la famille $\mathcal{H}$. Cette condition est parfois possible à réaliser lorsqu'on a une bonne connaissance du modèle sous-jacent à la probabilité P. Par exemple, si on sait à l'avance que le vecteur aléatoire d'entrée X est lié au vecteur aléatoire de sortie Y par une relation linéaire de type $Y = AX$, alors il suffit de restreindre la classe $\mathcal{H}$ à l'ensemble des applications linéaires de $\mathcal{Y}$ dans $\mathcal{X}$. L'erreur d'approximation est ainsi nulle.

Une telle situation peut arriver dans des situations plus générales. On peut par exemple considérer un modèle linéaire du type $Y = AX + B$ où B représente une perturbation aléatoire des données de moyenne nulle.

De façon générale, plus on est capable d'intégrer d'information a priori précises sur le modèle qui lie X et Y, plus on va être capable de réduire l'erreur d'approximation et la dimension d de la famille $\mathcal{H}$. C'est donc un vrai problème de modélisation mathématique qui nécessite souvent beaucoup de compétence métier. Par exemple, un météorologue peut être guidé par les équations de Navier-Stokes pour lier l'évolution du climat à des mesures de températures et de pression en quelques points de l'atmosphère. Cet aspect du travail est probablement un des plus intéressants ou au moins discriminants pour les "data-scientists".

2.2.2 Approximations linéaires et largeurs de Kolmogorov

Une méthode très populaire pour approcher l'application h^* consiste à utiliser une approximation linéaire. On se donne une famille $(\psi_d)_{1 \leq d \leq D}$ de fonctions élémentaires. Des choix typiques sont :

— Les polynômes ou les polynômes trigonométriques.
— Les splines.
— Les ondelettes.

On va alors chercher à approcher h^* par une combinaison linéaire de la forme

$$h^* \simeq \sum_{d=1}^{D} w_d \psi_d.$$

A quelle condition sur h^* est-ce qu'une telle approximation va être précise ? Une manière de répondre à cette question est d'uiliser la théorie des largeurs de Kolmogorov ou N-width. Je renvoie les étudiants intéressés au livre [7] pour plus de détails.

Supposons qu'on sache à l'avance que h^* appartient à une famille $\mathcal{F}$ (possiblement infinie) de fonctions provenant d'un espace vectoriel $(\mathcal{E}, \|\cdot\|)$ normé. Par exemple, on peut considérer l'espace $\mathcal{E} = L^2([0,1])$ muni de la norme $L^2 : \|\cdot\| = \|\cdot\|_{L^2}$. Pour l'espace $\mathcal{F}$, on peut considérer une famille de fonctions plus régulières telle que les fonctions de Sobolev W_p^r de norme inférieure à 1 pour un entier $r \in \mathbb{N}$. La largeur de Kolmogorov permet de déterminer :

— A quel point la famille $\mathcal{F}$ peut-être bien approchée par un sous-espace de dimension finie de fonctions $\mathcal{H}_D = \text{span}(\psi_1, \ldots, \psi_D)$.
— Une analyse précise permet aussi de déterminer une famille de fonctions (ψ_d) optimale.

Définition 2.1 (Largeur de Kolmogorov) *La largeur de Kolmogorov de dimension D (ou d-width) d'une famille $\mathcal{F}$ est définie par*

$$\delta_D(\mathcal{F}, \|\cdot\|) = \inf_{\dim(\mathcal{H}_D)=D} \sup_{f\in\mathcal{F}} \inf_{h\in\mathcal{H}_D} \|f - h\|. \tag{2.1}$$

Ici, il faut comprendre que l'optimisation est réalisée sur l'ensemble des sous-espaces vectoriels de dimension D.

Cette largeur correspond à la pire erreur d'approximation d'une fonction f de la classe $\mathcal{F}$ par un sous-espace vectoriel de dimension D. La base $(\psi_d)_{1\leq d\leq D}$ qui réalise l'infinimum est une base optimale d'approximation. Une illustration de ce concept est donnée sur la figure 2.1.

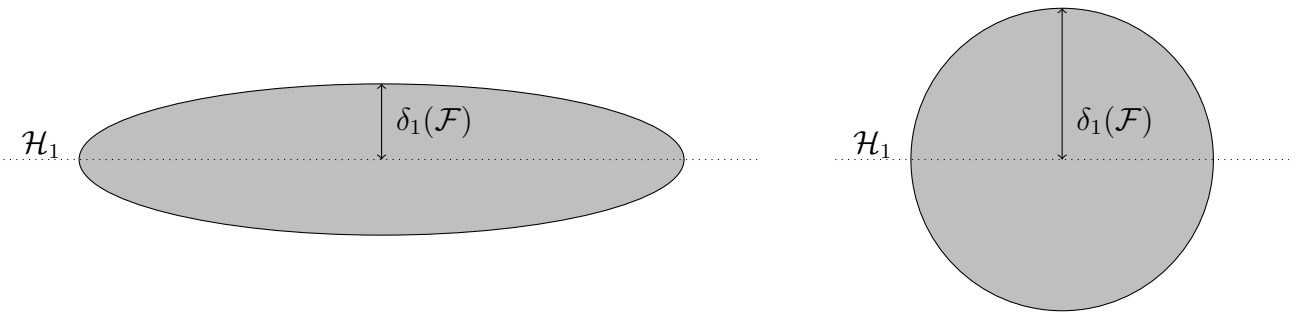

FIGURE 2.1 – Largeur de Kolmogorov d'ellipses en dimension 2. Une ellipse allongée est mieux approchée par un sous-espace linéaire de dimension 1 qu'un boule (la pire des familles). La largeur de Kolmogorov capture donc la courbure de familles de fonctions.

Proposition 2.1 *La largeur de Kolmogorov possède notamment les propriétés suivantes.*
- *Pour toute famille $\mathcal{F}$, $\delta_D(\mathcal{F})$ est une fonction décroissante en D.*
- *Pour tout scalaire α, on a $\delta_D(\alpha\mathcal{F}) = |\alpha|\delta_D(\mathcal{F})$.*
- *On a $\delta_D(\mathcal{F}) = \delta_D(\text{conv}(\mathcal{F}))$ où $\text{conv}(\mathcal{F})$ est l'enveloppe convexe de $\mathcal{F}$.*
- *$\mathcal{F}$ est compacte si et seulement si $\delta_D(\mathcal{F})$ tend vers 0 et que $\mathcal{F}$ est bornée.*

On peut alors s'intéresser au calcul de largeurs de Kolmogorov de certains ensembles. Une attention particulière a été portée aux espaces de Sobolev. Nous donnons ci-dessous un exemple pour des fonctions unidimensionnelles.

Définition 2.2 (Espaces de Sobolev) *L'espace de Sobolev $W_p^r([0,1])$ est défini par*

$$W_p^r = \{h \in C^{(r-1)}([0,1]), h^{r-1} \text{ absolûment continue}, h^{(r)} \in L^p([0,1])\}.$$

La boule unité associée est définie par

$$B_p^r = \{f \in W_p^r([0,1]), \|f^{(r)}\|_p \leq 1\}.$$

Théorème 2.1 (Espaces de Sobolev en dimension 1 [7]) *Le symbole $\equiv$ ci-dessous signifie "équivalent". La largeur de Kolmogorov des espaces de Sobolev satisfait :*

$$\delta_N(B_p^r, \|\cdot\|_p) \equiv n^{-r} \text{ si } 1 \leq q \leq p \leq +\infty. \tag{2.2}$$

On voit ici que les espaces de fonctions régulières (avec beaucoup de dérivées), peuvent être bien approchés par des sous-espaces de fonctions de petite dimension. De façon informelle, les bases fonctions (ψ_d) optimales pour ces espaces sont les bases d'ondelettes.

En dimension supérieure, l'analyse devient plus difficile et nous nous contenterons d'un résultat pour les espaces de Hilbert et pour la métrique L^2.

Théorème 2.2 (Espaces de Sobolev en dimension supérieure) *Soit $H^r = W_2^r([0,1]^p)$ et $B_p^r = \{f \in H^r, \|f^{(r)}\|_{L^2} \leq 1\}$. Le résultat suivant est valide :*

$$\delta_D(B_p^r, \|\cdot\|_2) = O(D^{-r/p}). \tag{2.3}$$

Ainsi, il faut de l'ordre de $D = \left(\frac{1}{\epsilon}\right)^{p/r}$ échantillons pour obtenir une approximation de précision ϵ d'une fonction dans $H^{(r)}$. Dans de nombreux problèmes, la dimension p est gigantesque (e.g. nombre de pixels d'une image, nombre de gènes d'un humain). Il est alors impossible d'approcher une fonction par une combinaison linéaire à part pour des fonctions très régulières (r grand). On parle de fléau de la dimension (curse of dimensionality).

A retenir

Les modèles linéaires sont capables de bien capturer les fonctions lisses en petite dimension. Ils sont souvent insuffisants lorsque les observations x et y vivent en grande dimension ou lorsque les fonctions de prédicitions sont peu régulières.

2.2.3 No free lunch theorems

Le théorème formalise le fait qu'il est impossible d'apprendre quoi que ce soit sans hypothèse a priori sur la loi qui relie les entrées x aux sorties y.

Théorème 2.3 (No free lunch theorem (e.g. Théorème 7.1 [8])) *Soit $\mathcal{X}$ un espace de dimension infinie et $\mathcal{Y}$ l'espace discret $\{0,1\}$. Soit $\mathcal{P}$ l'ensemble des probabilités sur $\mathcal{X}\times\mathcal{Y}$. Ce cadre correspond au problème le plus simple de classification binaire. Soit $\mathcal{D}_N = ((x_1,y_1),\dots,(x_N,y_N))$ un ensemble de N observations.*

Pour tout $N \geq 0$ et tout algorithme d'apprentissage $\mathcal{A}(\mathcal{D}_N) : \mathcal{X} \to \mathcal{Y}$, on a

$$\sup_{dp\in\mathcal{P}} \int (|\mathcal{A}(\mathcal{D}_N)(x) - y|) dp(x,y) \geq \frac{1}{2}.$$

Ce théorème indique que quelque soit l'algorithme d'apprentissage, il existera des lois liant x à y qui ne peuvent pas être apprises à partir d'un nombre *fini* d'observations. En effet, une erreur de 0.5 est celle produite par un algorithme renvoyant un nombre aléatoire dans $\{0,1\}$. Si on ne peut pas dépasser la performance d'un tel algorithme, c'est qu'on ne peut rien apprendre !

2.2.4 Expressivité des réseaux de neurones

De nombreux résultats pratiques et théoriques obtenus récemment convergent vers la même idée : les réseaux de neurones profonds peuvent exprimer efficacement bien plus de fonctions que les approximations linéaires. Un premier résultat en ce sens a été obtenu en 1989 par G. Cybenko, puis affiné par des auteurs tels que Allan Pinkus (en référence). Nous donnons ci-dessous un de ses résultats.

Théorème 2.4 (Approximation universelle) *Soit $f : \mathbb{R}^p \to \mathbb{R}^q$ une fonction continue arbitraire. Soit $K \subset \mathbb{R}^p$ un ensemble compact et $\epsilon > 0$ une précision arbitraire. Soit σ une fonction d'activation* qui n'est pas un polynôme.

Alors il existe une fonction f_ϵ de la forme (un réseau à deux couches) :

$$f_\epsilon = W_2 \circ \sigma \circ W_1 \tag{2.4}$$

où W_2 et W_1 sont des applications affines, telles que

$$\sup_{x\in K} \|f(x) - f_\epsilon(x)\| < \epsilon. \tag{2.5}$$

Exercice 2.2.1 *Construire un réseau de neurones de $\mathbb{R}$ dans $\mathbb{R}$ avec des fonctions d'activation ReLU et comprendre sa structure fonctionnelle. Le comparer à des splines d'ordre 1 à noeuds fixes.*

Ce terme indique que n'importe quelle fonction peut être approchée par un réseau de neurones à deux couches. Cependant, il ne dit rien sur la taille du réseau : l'application W_1 va de $\mathbb{R}^p$ dans $\mathbb{R}^m$ avec $m \in \mathbb{N}$ arbitrairement grand.

Récemment, de nombreux travaux essaient de caractériser la taille et la profondeur des réseaux de neurones nécessaires pour approcher des classes de fonctions données. Un lecteur intéressé est renvoyé au papier suivant par exemple [9]. De façon très grossière, les conclusions qui sont apportées sont les suivantes :

A retenir

- Ce qui peut être fait par des approximations linéaires peut aussi l'être avec des réseaux de neurons, avec un nombre de paramètres du même ordre de grandeur.
- Les réseaux de neurones permettent aussi de reproduire des fonctions que les approximations linéaires sont incapables de reproduire à nombre de paramètres constant.
- Pour certaines applications, les réseaux de neurones permettent de casser le fléau de la dimension (i.e. de nécessiter un nombre de paramètres d non exponentiel dans la dimension p des entrées).
- Le coût à payer pour les réseaux de neurones n'est pas négligeable : la non convexité des énergies à minimiser, le manque de certifiabilité des réseaux résultants, le choix difficile d'une structure de réseau souvent obtenue par essai/erreur.

2.3 Erreurs d'estimation

2.3.1 Un contrôle uniforme

La théorie de Vapnik-Chervonenkis [10] permet de mesurer la complexité d'une classe de fonctions $\mathcal{H}$ à travers ce qui est désormais appelé la dimension de Vapnik-Chervonenkis ou dimension VC. Nous n'allons pas rentrer dans cette théorie ici, mais simplement donner une de ses conséquences pour le contrôle du risque empirique dans le cadre d'estimateurs linéaires.

Théorème 2.5 (Contrôle d'erreur uniforme) *Supposons que la famille $\mathcal{H}$ soit paramétrée* linéairement *par le paramètre $w \in \mathbb{R}^D$. Dans ce cas, l'inégalité suivante est vérifiée :*

$$\sup_{h \in \mathcal{H}} |E(h) - E_N(h)| \leq c\sqrt{\frac{D}{N}}, \tag{2.6}$$

où c est une constante universelle et où l'espérance est évaluée par rapport à la loi P. Ceci implique en particulier que

$$\mathcal{E}_{est} \leq 2c\sqrt{\frac{D}{N}}.$$

Pour une famille $\mathcal{H}$ arbitraire, on peut obtenir le résultat suivant [1] :

1. un étudiant intéréssé peut regarder l'excellent livre de R. Vershynin [11] et en particulier le chapitre 8

Théorème 2.6 (Contrôle d'erreur uniforme) *Supposons que $Y = T(X)$ pour une certaine transformation T déterministe. Soient $X_1, \dots, X_N$ des variables aléatoires indépendantes distribuées suivant la loi P. Alors l'inégalité suivante est vérifiée :*

$$\sup_{h \in \mathcal{H}} |E(f) - E_N(f)| \leq c\sqrt{\frac{vc(\mathcal{H})}{N}}, \tag{2.7}$$

où c est une constante universelle et où $vc(\mathcal{H})$ est la dimension de Vapnik-Chervonenkis de $\mathcal{H}$.

2.3.2 Contrôles plus avancés

La borne d'erreur (2.6) est en général assez pessimiste. Le taux de convergence en $\sqrt{\frac{D}{N}}$ peut être amélioré sous des hypothèses plus restrictives mais raisonnables. De façon générale, on peut espérer obtenir des taux du type :

$$\mathcal{E}_{est} = O\left((D/N \log(D/N))^{\alpha}\right),$$

où α est un paramètre dans l'intervalle $[1/2, 1]$. Ce paramètre α dépend de la fonction perte utilisée et peut typiquement valoir 1 pour une fonction perte quadratique.

2.4 Conclusion

A retenir

Le taux de convergence du risque empirique par rapport au nombres de données est très lent (du type $1/N^{\alpha}$ avec α petit). Il ne sert donc à rien d'utiliser des algorithmes d'optimisation très précis et coûteux (à moins d'avoir des bases de données gigantesques ou infinies), car ils n'apporteraient rien, sinon de l'over-fitting. Nous allons voir qu'un choix de prédilection dans ce cadre est le gradient stochastique et ses variantes.
Les théories que je viens de présenter sont presque ce qui se fait de plus avancé avant 2015. Elles indiquent qu'il est nécessaire d'observer $n > d$ observations pour estimer un modèle, sans quoi l'erreur d'estimation n'est pas contrôlable. Ceci rentre en contradiction avec ce qui est observé actuellement, où des réseaux de neurones gigantesques $D \gg N$ donnent des performances bien meilleures que des modèles plus réduits.
Nous verrons plus tard dans ce cours des avancées récentes qui semblent prometteuses pour expliquer ce phénomène.

2.5 Les modèles sur-paramétrés

Cette section est fortement inspirée du papier récent [12], que je trouve réellement enthousiasmant.

Les statistiques "traditionnelles", i.e. étudiées et enseignées jusque dans les années 2010 (et encore aujourd'hui) ont eu tendance à rejeter les modèles complexes de dimension $D \simeq N$ ou même $D \gg N$. Dans ce régime, le modèle est tellement riche, qu'il peut *interpoler* exactement les données. C'est-à-dire assurer que $h(x_n, w) = y_n$ pour tout n et un certain vecteur de poids w. Par exemple, on peut trouver la phrase suivante dans un des livres de référence du domaine [13] (que je ne recommande pas pour ce cours) :

Remise en question : *A model with zero training error is overfit to the training data and will typically generalize poorly.*

En somme : interpoler fait courir le risque de sur-ajustement, qui serait mauvais pour des données extérieures au jeu d'entraînement.

La pratique récente montre que cette idée est douteuse : les meilleurs modèles pour des tâches spécifiques peuvent contenir des milliards de paramètres. Par exemple, le réseau GPT3 de traitement du langage contient $D = 175$ milliars de paramètres. Il peut générer des articles de presse automatiquement, très difficilement distinguables de ceux générés par des êtres humains.

A retenir

a théorie "traditionnelle" en statistique est précise pour des modèles simples (linéaires notamment). Elle peut être très pessimiste dans le cas non linéaire des réseaux de neurones. Une observation récente est que certains modèles sur-paramétrés peuvent être appris avec peu de données (par exemple moins de données que le nombre de paramètres).

Chapitre 3

Gradient stochastique et variantes

Dans l'introduction, nous avons vu que de nombreuses tâches d'apprentissage consistaient à résoudre des problèmes d'optimisation de la forme

$$\min_{w\in\mathbb{R}^D} \frac{1}{N}\sum_{n=1}^{N} f_n(w) \text{ où } f_n(w) = \ell(h(x_n, w), y_n). \tag{3.1}$$

Chaque terme dans l'énergie f ne contient qu'un seul exemple de la base d'apprentissage. Si la fonction perte et la fonction $h(x,\cdot)$ est différentiable par morceaux, on peut simplement appliquer une descente de gradient pour résoudre (3.1) :

$$w_{k+1} = w_k - \alpha_k \nabla f(w_k) = w_k - \frac{\alpha_k}{N}\sum_{n=1}^{N} \nabla f_n(w_k),$$

où α_k est un pas de descente qui dépend de la régularité de f. La faiblesse de cette approche, notamment lorsque le nombre de données N est gigantesque est que chaque itération requiert de faire une passe sur l'intégralité des données.

De plus dans certaines applications, le nombre de données peut être infini. On peut par exemple penser à un flux d'information arrivant continuellement sur le web. Dans ce cas, le gradient ne peut même plus être calculé numériquement... Comment faire face à ces difficultés ?

3.1 Le gradient stochastique

Dans ce paragraphe, nous introduisons le gradient stochastique et certains des termes usuels utilisés dans ce domaine, tels que batch, epoch, learning rate.

3.1.1 Le principe

La version la plus simple du gradient stochastique consiste à ne pas utiliser le gradient complet, mais de choisir au hasard des indices n et de suivre uniquement le gradient de cette coordonnée. Ceci donne lieu à l'algorithme 1.

Algorithm 1: Une version élémentaire de gradient stochastique.

Input:
w_0 : un point de départ.
for $k = 0, \ldots,$ **do**
  Choisir un pas $\alpha_k > 0$ (aussi appelé learning rate)
  Choisir un indice $i_k \in \{1, \ldots, N\}$ au hasard.
  Calculer $w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k)$

Le terme stochastique est lié au fait qu'on tire les indices au hasard. Un intérêt évident de cette approche est que le coût par itération est N fois plus petit que le coût initial. Dans la suite de ce cours, on va essayer de comprendre ce qu'on perd en utilisant une unique composante et si le compromis entre les gains et les pertes est acceptable.

Remarque 3.1 (Descente de gradient ?) *Une première remarque importante est que l'algorithme du gradient stochastique n'est pas un algorithme de descente. Par exemple, la fonction scalaire $f(w) = f_1(w) + f_2(w) = w^2$ avec $f_1(w) = 2w^2$ et $f_2(w) = -w^2$ est convexe. Par contre un pas suivant la seconde composante mène toujours à une augmentation de la fonction objectif. Ainsi, il est erroné en général de parler de descente de gradient stochastique.*

Une alternative est de ne pas tirer un unique élément n_k, mais plutôt un sous-ensemble d'indices $I_k \subset \{1, \ldots, N\}$. On parle alors de *batch stochastic gradient* où le batch est l'ensemble I_k. Ceci mène à l'itération

$$w_{k+1} = w_k - \frac{\alpha_k}{|I_k|} \sum_{i \in I_k} \nabla f_i(w_k).$$

Cette version couvre le gradient usuel si $I_k = \{1, \ldots, N\}$ et le gradient stochastique si I_k est un singleton.

Dans la littérature, on parle d'une *epoch* lorsque l'algorithme a fait une passe suivant N composantes. Un des objectifs est – pour un nombre d'epoch donné – d'obtenir la meilleure réduction de la fonction objectif. Une descente de gradient usuelle effectue une epoch en 1 itération tandis qu'un gradient stochastique réalise une epoch en n itérations.

A retenir

Nous avons introduit les termes suivants :

Gradient stochastique : éléments de la somme tirés au hasard.

Learning rate : longueur du pas dans la direction opposée au gradient stochastique.

Batch : ensemble des indices tirés au hasard.

Epoch : nombre de passes efffectués sur la totalité des données disponibles.

3.1.2 Garanties théoriques dans le cas fortement convexe

Nous étudions ici les taux de convergence de cette méthode dans le cas convexe.

Hypothèses et premiers résultats

Dans tous les cas, nous aurons besoin de l'hypothèse suivante.

Hypothèse 3.1 *Les fonctions élémentaires f_n sont $C^1(\mathbb{R}^D)$ et la fonction objectif f est $C^{1,L}(\mathbb{R}^D)$, i.e.*

$$\|\nabla f(w) - \nabla f(v)\|_2 \leq L\|w - v\|_2$$

pour toute paire (v, w). On suppose de plus que f est bornée inférieurement par une valeur $f_{min} > 0$.

Dans la suite on utilise la notation $\mathbb{E}_{i_k}$ pour définir une espérance par rapport à l'indice aléatoire i_k.

Proposition 3.1 *Sous l'hypothèse 3.1, l'inégalité suivante est vérifiée :*

$$\mathbb{E}_{i_k}(f(w_{k+1})) \leq f(w_k) - \alpha_k \langle \nabla f(w_k), \mathbb{E}_{i_k}(\nabla f_{i_k}(w_k)) \rangle + \frac{L\alpha_k^2}{2} \mathbb{E}_{i_k}(\|\nabla f_{i_k}(w_k)\|_2^2).$$

Preuve. La notation $\mathbb{E}_{i_k}$ indique ici qu'on calcule l'espérance par rapport à la variable aléatoire i_k. On sait (voir le cours d'optimisation convexe) que pour tout $h \in \mathbb{R}^d$:

$$f(w_k + h) \leq f(w_k) + \langle \nabla f(w_k), h \rangle + \frac{L}{2}\|h\|_2^2. \tag{3.2}$$

En particulier pour $h = -\alpha_k \nabla f_{i_k}(w_k)$, on obtient

$$f(w_{k+1}) \leq f(w_k) - \alpha_k \langle \nabla f(w_k), \nabla f_{i_k}(w_k) \rangle + \frac{L}{2}\|\nabla f_{i_k}(w_k)\|_2^2. \tag{3.3}$$

Il faut noter que ci-dessus, $f(w_{k+1})$ est une variable aléatoire, puisque i_k est lui-même aléatoire. C'est pourquoi on peut calculer l'espérance par rapport à la variable aléatoire i_k, ce qui donne le résultat annoncé. □

Cette inégalité donne des garanties très faibles à chaque itération. Pour obtenir plus, nous avons besoin d'hypothèses supplémentaires.

Hypothèse 3.2 (Hypothèses sur le gradient stochastique) *A chaque itération de l'algorithme 1, nous supposons que les conditions suivantes sont satisfaites :*

1. *Les indices aléatoires i_k sont tirés indépendamment les uns des autres.*
2. $\mathbb{E}_{i_k}[\nabla f_{i_k}(w_k)] = \nabla f(w_k)$
3. $\mathrm{Var}(\|\nabla f_{i_k}(w_k)\|_2) = \mathbb{E}_{i_k}[\|\nabla f_{i_k}(w_k)\|_2^2] - \|\nabla f(w_k)\|_2^2 \leq \sigma^2$ *pour* $\sigma^2 \geq 0$.

La deuxième hypothèse indique ∇f_{i_k} est un estimateur non biaisé de ∇f. La troisième propriété permet de contrôler la variance de la norme du gradient stochastique. Cette variance indique à quel point les gradients des différentes fonctions f_n sont alignés. En un sens, cette variance mesure la complexité de la tâche d'apprentissage. Plus elle est grande, moins les gradients sont alignés, ce qui significe que chaque donnée a tendance à tirer le modèle dans des directions antagonistes. Au contraire, si tous les gradients sont alignés, la minimisation d'un unique f_n est presque équialente à la minimisation de la somme.

L'exemple de choix le plus populaire qui satisfait ces hypothèses est le suivant.

Exemple 3.1.1 (Échantillonnage uniforme) *On suppose que les indices i_k sont tirés indépendamment, uniformément dans l'intervalle $\{1, \dots, N\}$. Alors les hypothèses 3.2 sont satisfaites.*

Nous pouvons maintenant obtenir des garanties plus fortes.

Proposition 3.2 *Sous les hypothèses 3.1 et 3.2, l'inégalité suivante est vérifiée pour tout k :*

$$\mathbb{E}_{i_k}[f(w_{k+1}) - f(w_k)] \leq -\left(\alpha_k - \frac{L\alpha_k^2}{2}\right)\|\nabla f(w_k)\|_2^2 + \frac{L\alpha_k^2}{2}\sigma^2 \tag{3.4}$$

On voit donc que si le pas α_k est suffisamment petit, la fonction coût décroît au fur et à mesure des itérations en espérance. Ce résultat est central pour déterminer des taux de convergence du gradient stochastique sous différentes conditions. Nous en présentons certaines ci-dessous.

Nous allons maintenant travailler avec l'hypothèse supplémentaire suivante :

Hypothèse 3.3 (Forte convexité) *Il existe un paramètre $\mu > 0$ tel que f est μ-fortement convexe, i.e.*

$$f(x) \geq f(w) + \langle \nabla f(w), (x - w)\rangle + \frac{\mu}{2}\|x - w\|_2^2. \tag{3.5}$$

Cette hypothèse implique en particulier que la fonction f admet un minimiseur w^* unique. On note $f^* = f(w^*)$.

Convergence à pas constants

Proposition 3.3 *Sous les hypothèses 3.1 et 3.3, l'inégalité suivante est vérifiée pour tout w :*

$$\|\nabla f(w)\|_2^2 \geq 2\mu(f(w) - f^*) \tag{3.6}$$

Preuve. D'après l'inégalité 3.5, on a

$$\begin{aligned} f(x) &\geq f(w) + \langle \nabla f(w), (x - w)\rangle + \frac{\mu}{2}\|x - w\|_2^2 \\ &\geq \min_{x'} f(w) + \langle \nabla f(w), (x' - w)\rangle + \frac{\mu}{2}\|x' - w\|_2^2 \\ &= f(w) - \frac{1}{2\mu}\|\nabla f(w)\|_2^2, \end{aligned}$$

où la dernière inégalité est obtenue en se rendant compte que le minimiseur est $x = w - \nabla f(w)/\mu$. En choisissant $x = w^*$ on obtient l'inégalité annoncée. Cette inégalité indique simplement que pour une fonction fortement convexe, le gradient est nul au minimiseur. Sa norme croît forcément en s'éloignant du minimiseur, puisque la plus petite valeur propre de la Hessienne est bornée inférieurement par μ. □

Le premier résultat de convergence est le suivant.

Théorème 3.1 (Convergence dans le cas fortement convexe) *Sous les hypothèses 3.1, 3.3 et 3.2 et lorsqu'on choisit un pas constant $\alpha_k = \alpha \in \left]0, \frac{1}{L}\right]$, l'algorithme du gradient stochastique assure que :*

$$\mathbb{E}[f(w_k) - f^*] \leq \frac{\alpha L \sigma^2}{2\mu} + (1 - \alpha\mu)^k \left[f(w_0) - f^* - \frac{\alpha L \sigma^2}{2\mu} \right]. \tag{3.7}$$

Preuve. En appliquant la proposition 3.2, on obtient :

$$\begin{aligned}\mathbb{E}_{i_k}[f(w_{k+1}) - f(w_k)] &\leq -\left(\alpha - \frac{L\alpha^2}{2}\right) \|\nabla f(w_k)\|_2^2 + \frac{L\alpha^2\sigma^2}{2} \\ &\overset{Prop.3.3}{\leq} -2\mu\left(\alpha - \frac{L\alpha^2}{2}\right)(f(w_k) - f^*) + \frac{L\alpha^2\sigma^2}{2}.\end{aligned}$$

En exploitant le fait que $\alpha \leq \frac{1}{L}$, on obtient :

$$\mathbb{E}_{i_k}[f(w_{k+1}) - f(w_k)] \leq -\mu\alpha(f(w_k) - f^*) + \frac{L\alpha^2\sigma^2}{2}.$$

On remarque maintenant que $\mathbb{E}_{i_k}[f^* - f(w_k)] = f^* - f(w_k)$, ce qui permet d'ajouter et de soustraire f^* :

$$\mathbb{E}_{i_k}[f(w_{k+1}) - f^*] + f^* - f(w_k) \leq -\mu\alpha(f(w_k) - f^*) + \frac{L\alpha^2\sigma^2}{2}$$

Soit encore

$$\mathbb{E}_{i_k}[f(w_{k+1}) - f^*] \leq (1 - \mu\alpha)(f(w_k) - f^*) + \frac{L\alpha^2\sigma^2}{2}.$$

Comme $\mu \leq L$, on a $\frac{1}{L} \leq \frac{1}{\mu}$ et donc $1 - \mu\alpha \in]0, 1[$. On peut finir en soustrayant $\frac{L\alpha\sigma^2}{2\mu}$ de chaque côté :

$$\begin{aligned}\mathbb{E}_{i_k}[f(w_{k+1}) - f^*] - \frac{L\alpha\sigma^2}{2\mu} &\leq (1 - \mu\alpha(f(w_k) - f^*)) + \frac{L\alpha^2\sigma^2}{2} - \frac{L\alpha\sigma^2}{2\mu} \\ &= (1 - \mu\alpha)\left[f(w_k) - f^* - \frac{L\alpha\sigma^2}{2\mu} \right]\end{aligned} \tag{3.8}$$

Comme les indices aléatoires i_k sont indépendants 2 à 2, on a

$$\mathbb{E}[f(w_k)] = \mathbb{E}_{i_0}[\mathbb{E}_{i_1}[\ldots \mathbb{E}_{i_{k-1}}(f(w_k))]]. \tag{3.9}$$

On peut alors utiliser l'inégalité (3.8) récursivement pour obtenir le résultat annoncé. □

Remarque 3.2 (Comparaison à une descente de gradient ordinaire) *Pour mieux comprendre le théorème 3.1, il est important de le comparer à ce que donnerait une descent de gradient ordinaire :*

$$f(w_k) - f^* \leq (1 - \mu\alpha)^k [f(w_0) - f^*]. \tag{3.10}$$

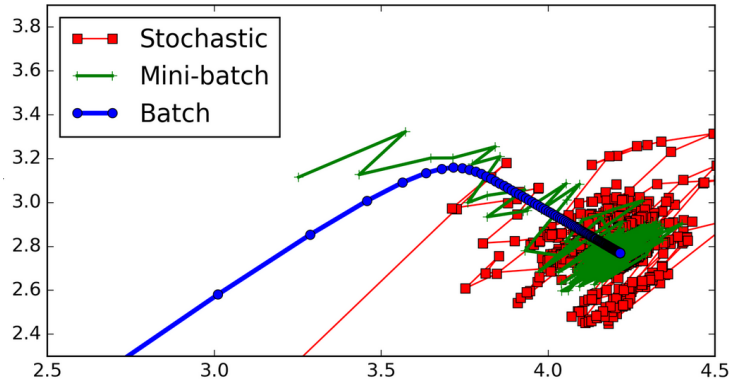

FIGURE 3.1 – Comparaison d'une descente de gradient standard (batch), stochastique (un élément par itération), mini-batch (quelques éléments par itération). On observe que la taille du voisinage autour du minimiseur diminue en fonction de la taille du batch.

Ce taux est similaire au terme de droite dans l'inégalité (3.7)*. Une différence importante est le terme de gauche : la descente de gradient stochastique à pas constant ne permet pas de minimiser exactement l'énergie. En pratique, elle converge seulement dans un* voisinage *de la valeur optimale f^*. La raison tient au fait que le gradient n'est pas exact : ainsi une fois proche du minimiseur, on tourne autour de façon aléatoire sans jamais y arriver. Ce phénomène est illustré sur la figure 3.1.*

Convergence avec pas décroissants

On peut diminuer le terme de variance $\frac{\alpha L \sigma^2}{2\mu}$ en baissant le pas. Cependant, le taux de convergence devient plus lent. Une stratégie pratique pour le réduire est d'utiliser un pas décroissant. On commence par fixer un pas assez grand jusqu'à ce que la suite $f(w_k)$ semble stagner (suivant un certain critère). Lorsque c'est le cas, on divise le pas par un facteur 2 et on reprend la méthode. Si ce processus est bien effectué, on peut "tuer" le terme de variance asymptotiquement. Cependant, on peut montrer que quelle que soit la stratégie, le taux de convergence est sous-linéaire, avec au mieux un taux du type :

$$\mathbb{E}[f(w_k) - f^*] = O\left(\frac{1}{k}\right). \tag{3.11}$$

De façon générale, on peut choisir des pas qui satisfont les conditions

$$\sum_{k=0}^{\infty} \alpha_k = +\infty \quad \text{et} \quad \sum_{k=0}^{\infty} \alpha_k^2 < +\infty.$$

Le taux optimal (en théorie) peut être obtenu avec des pas décroissants.

Théorème 3.2 (Convergence pour des pas décroissants) *Sous les hypothèses 3.1, 3.3 et 3.2 et avec un pas*

$$\alpha_k = \frac{\beta}{k+\gamma} \text{ où } \beta \geq \frac{1}{\mu} \text{ et } \alpha_0 = \frac{\beta}{\gamma} \leq \frac{1}{L}, \text{ on obtient :}$$

$$\mathbb{E}[f(w_k) - f^*] \leq \frac{\nu}{\gamma + k}, \tag{3.12}$$

où

$$\nu = \max\left(\gamma(f(w_0) - f^*), \frac{\beta^2 L\sigma^2}{2(\beta\mu - 1)}\right).$$

Preuve. □

Un taux de convergence en $O(1/k)$ est sous-linéaire. Il est identique à un taux de convergence d'une descente de gradient ordinaire sans l'hypothèse de forte convexité. Du point de vue de l'optimiseur, c'est donc probablement une mauvause méthode : elle fait passer d'un taux de convergence linéaire (exponentiel) à un taux sous-linéaire. Il faut cependant se rappeler que le taux de convergence des estimateurs est lui aussi sous-linéaire et qu'il ne sert donc à rien d'obtenir des taux trop rapides.

Remarque 3.3 *On voit dans les deux théorèmes précédents que la performance des méthodes dépend de la connaissance des paramètres L et μ. Ceux-ci sont en général difficiles à estimer et dépendent des données. En pratique, on est donc amener à les estimer. Ceci peut-être effectué avec des recherche sur grille (on teste plusieurs paramètres pour obtenir un bon taux). On peut aussi les estimer en tirant quelques données au hasard pour se faire une idée grossière de leur ordre de grandeur. On verra plus tard qu'il est possible de faire des recherches linéaires de pas.*

A retenir

- Dans le cas fortement convexe, le gradient stochastique à pas constant convergence seulement dans un voisinage du minimiseur.
- Il faut utiliser des pas décroissants pour réduire la taille de ce voisinage.
- Le prix à payer est uine convergence sous-linéaire alors qu'un gradient ordinaire convergerait linéairement.
- Ce prix est acceptable si on prend en compte l'erreur d'estimation.

3.1.3 Le cas non convexe

Les descentes de gradient stochastiques sont l'outil principal pour l'entraînement de réseaux de neurones. Ceux-ci mènent systématiquement à des fonctions objectif non convexes. Il est donc important d'analyser comment se comporte le gradient stochastique dans ce cas. Dans cette partie, nous nous intéressons à son comportement local : à quelle vitesse est-ce que le gradient stochastique converge vers un point critique, c'est-à-dire un point $w \in \mathbb{R}^d$ tel que $\nabla f(w) = 0$.

Pour commencer, rappelons qu'une descente de gradient usuelle assure le résultat suivant :

$$\min_{0 \leq k \leq K-1} \|\nabla f(w_k)\|_2 = O\left(\frac{1}{\sqrt{K}}\right).$$

Ainsi, il existe un point le long de la trajectoire (pas forcément le dernier) qui a un gradient faible. Qu'en est-il du gradient stochastique ?

Théorème 3.3 *Supposons que les hypothèses 3.1 et 3.2 soient satisfaites et qu'on utilise un pas constant* $0 < \alpha \leq \frac{1}{L}$. *Dans ce cas, on a*

$$\mathbb{E}\left[\frac{1}{K}\sum_{k=0}^{K-1}\|\nabla f(w_k)\|_2^2\right] \leq \frac{\alpha L\sigma^2}{2} + \frac{2(f(w_0)-f^*)}{\alpha K}. \tag{3.13}$$

Preuve. D'après la proposition 3.2, on a

$$\begin{aligned}\mathbb{E}_{i_k}[f(w_{k+1}) - f(w_k)] &\leq -\left(\alpha - \frac{\alpha^2 L}{2}\right)\|\nabla f(w_k)\|_2^2 + \frac{\alpha^2 L\sigma^2}{2} \\ &\leq -\frac{\alpha}{2}\|\nabla f(w_k)\|_2^2 + \frac{\alpha^2 L\sigma^2}{2}.\end{aligned}$$

En prenant l'espérance par rapport à tous les indices, on obtient :

$$\mathbb{E}[f(w_{k+1}) - f(w_k)] \leq -\frac{\alpha}{2}\mathbb{E}[\|\nabla f(w_k)\|_2^2] + \frac{\alpha^2 L\sigma^2}{2}.$$

En sommant cette inégalité de $k = 0$ à $k = K-1$, on obtient

$$\mathbb{E}[f(w_K) - f(w_0)] \leq \frac{K\alpha^2 L\sigma^2}{2} - \frac{\alpha}{2}\sum_{k=0}^{K-1}\mathbb{E}[\|\nabla f(w_k)\|_2^2].$$

En utilisant le fait que $f(w) \geq f_{\min}$ pour tout w, on a de plus $\mathbb{E}[f(w_K) - f(w_0)] \geq f_{\min} - f(w_0)$. Ce qui conclut la preuve. □

De la même façon que dans le cas convexe, on voit que le caractère stochastique de la descente empêche la convergence vers un point critique. Cependant, la norme du gradient converge en moyenne vers une valeur bornée supérieurement par $\frac{\alpha L\sigma^2}{\mu}$. Celle-ci peut être rendue arbitrairement faible en choisissant un petit pas α. De la même façon que précédemment, on peut choisir une suite de pas décroissants pour obtenir une convergence du gradient vers 0. On peut donc donner la conclusion suivante :

A retenir

Le gradient stochastique avec pas décroissants converge vers des points critiques en espérance.

Remarque 3.4 *Avant d'aller plus loin, notons que c'est un résultat d'une portée limitée car :*

- *Un point critique n'est pas forcément un minimiseur, ca peut être un point selle (e.g.* 0 *est point critique de* $f(t) = t^3$*) ou même un maximum local. Ce phénomène est possible, mais reste très peu probable [14].*
- *Un aspect beaucoup plus gênant est qu'un minimum local peut être fortement sous-optimal par rapport au minimiseur global. Ce deuxième point fait actuellement l'objet de nombreuses recherches. Quel est le paysage energétique des énergies permettant d'optimiser des réseaux de neurones ? On commence à trouver des réponses partielles dans le cas des réseaux à deux couches [15] et – au contraire – asymptotiquement quand le nombre de couches tend vers l'infini [16].*

3.1.4 Notes finales

Avant d'aller plus loin, ajoutons quelques commentaires.

Conditionnement : Tous les résultats évoqués font intervenir la constante de Lipschitz L ou le conditionnement L/μ de la fonction f. Il est possible de réduire ces valeurs en changeant de métrique. Une solution simple consiste à considérer une constante de Lipschitz individuelle L_n pour chaque fonction f_n. Ceci permet d'équilibrer l'importance relative des différentes fonctions f_n et d'échantillonner plus fréquemment celles qui ont une grande constante de Lipschitz.

Précision des taux : les taux de convergence donnés sont ajustés, uniformément, sur la classe des fonctions Lipschitz. On peut trouver des fonctions f pour lesquelles l'algorithme du gradient stochastique ne convergera pas plus rapidement que les taux donnés. Ceci ne signifie pas qu'il ne peut pas bien mieux se comporter en pratique.

Cas non différentiable : La théorie donnée ci-dessus peut s'étendre au cas non fortement convexe et aussi non différentiable. Cependant, les preuves deviennent très difficiles et je ne souhaite donc pas rentrer dans le détail ici.

3.2 Accélérations et généralisations

Le taux de convergence du gradient stochastique naïf présenté précédemment est assez mauvais. De nombreux efforts ont donc été consentis pour l'accélérer et notamment - réduire la variance - le plus rapidement possible. Un autre aspect important est le préconditionnement heuristique de ces méthodes.

3.2.1 Réduction de variance

Mini-batch Une faiblesse du gradient stochastique est l'apparition des termes de variance de type $\frac{\alpha L\sigma^2}{2\mu}$ dans le théorème 3.1. On peut réduire la variance de l'estimateur du gradient en considérant des batchs plus grands. Supposons par exemple qu'à chaque itération, on tire non pas 1 indice aléatoire mais $n_b \in \mathbb{N}$. On peut montrer dans ce cas [1], qu'on obtient des taux de convergence de la forme :

$$\mathbb{E}[f(w_k) - f(w^*)] \leq \frac{\alpha L\sigma^2}{2\mu n_b} + (1-\alpha\mu)^k \left[f(w_0) - f^* - \frac{\alpha L\sigma^2}{2\mu n_b} \right]. \tag{3.14}$$

Le coût par itération des techniques de mini-batch est augmenté d'un facteur n_b, mais la variance est elle aussi réduite d'un facteur n_b. La grande majorité des algorithmes dans les librairies standards d'optimisation stochastique permettent de choisir des tailles de batch. Le choix d'une bonne taille me semble pour le moment plus relever de l'artisanat que d'une théorie solide. Une intuition de l'intérêt de cette méthode est donnée sur la figure 3.1.

1. Bon entraînement.

SVRG : Stochastic Variance Reduced Gradient Une autre manière assez simple de réduire la variance est de calculer le gradient complet après quelques itérations de gradient stochastique usuel. Dans l'Algorithme 2, un gradient complet est calculé toutes les m itérations. L'idée sous-jacente est que le gradient modifié

$$\tilde{g}_j = \nabla f(w_k) - \nabla f_{i_j}(w_k) + \nabla f_{i_j}(\tilde{w}_j)$$

possède une variance plus faible que le gradient stochastique $\nabla f_{i_j}(w_k)$. La complexité moyenne par itération de cet algorithme est plus grande, puisqu'il faut calculer le gradient complet toutes les m itérations. Cependant, cette stratégie permet d'améliorer le taux de convergence comme le montre le théorème suivant.

Théorème 3.4 *Si les hypothèses 3.1 et 3.3 sont satisfaites et si le pas α satisfait l'inégalité*

$$\rho = \frac{1}{1-2\alpha L}\left(\frac{1}{M\mu\alpha} + 2L\alpha\right) \in]0,1[,$$

alors

$$\mathbb{E}[f(w_k) - f^*] \leq \rho^k (f(w_0) - f^*). \tag{3.15}$$

On voit donc que le terme de variance en σ^2 a disparu.

Algorithm 2: La méthode SVRG.

Input:
w_1 : un point de départ. $\alpha > 0$, le pas. $M \in \mathbb{N}$.
for $k = 0, \ldots,$ **do**
  Calculer le gradient complet $\nabla f(w_k) = \frac{1}{N}\sum_{n=1}^{N} \nabla f_n(w_k)$.
  Poser $\tilde{w}_0 = w_k$. **for** $j = 1, \ldots, M$ **do**
    Choisir un indice $i_j \in \{1, \ldots, N\}$ au hasard.
    $\tilde{g}_j = \nabla f(w_k) - \nabla f_{i_j}(w_k) + \nabla f_{i_j}(\tilde{w}_j)$.
    $\tilde{w}_{j+1} = \tilde{w}_j - \alpha \tilde{g}_j$
  Option 1 : poser $w_{k+1} = \tilde{w}_{m+1}$
  Option 2 : poser $w_{k+1} = \frac{1}{M}\sum_{j=1}^{M} \tilde{w}_{j+1}$
  Option 3 : Tirer $j \in \{1, \ldots, M\}$ uniformément et poser $w_{k+1} = \tilde{w}_{j+1}$

A retenir

L'algorithme SVRG peut être vu comme un intermédiaire entre le gradient usuel et un gradient stochastique. Il peut se montrer efficace pour les applications qui nécessitent une bonne précision. Cependant, la convergence dans les premières epochs est en général moins bonne que celle d'un gradient stochastique usuel. Si on souhaite une bonne précision, il vaut probablement mieux utiliser un gradient accéléré (dans le cas convexe) pour obtenir de très bonne précisions.
A priori, cette méthode devrait donc relativement rarement être la méthode à adopter.

SAGA : Stochastic Average Gradient Algorithm La philosophie sous-jacente à la méthode SAGA est la même que celle de SVRG : réduire la variance du gradient stochastique. Pour atteindre cet objectif, la méthode SAGA effectue une estimation du gradient en utilisant le gradient stochastique calculé à N différents points du processus d'optimisation. La méthode est décrit dans l'Algorithme 3. Elle jouit des propriétés suivantes.

Théorème 3.5 *Si les hypothèses 3.1 et 3.3 sont vérifiées et si le pas* $\alpha = \frac{1}{2(\mu N+L)}$*, alors*

$$\mathbb{E}[f(w_k) - f^*] \leq \left(1 - \frac{\mu}{2(\mu N + L)}\right)^k \left(\|w_0 - w^*\|_2^2 + \frac{N(f(w_0) - f^*)}{\mu N + L}\right). \tag{3.16}$$

On peut montrer que les taux de convergence de SAGA et SVRG sont du même ordre de grandeur. L'intérêt de SAGA est qu'il ne requiert pas le calcul du gradient entier.

Algorithm 3: La méthode SAGA.

Input:
$w^{(0)}$: un point de départ. $\alpha > 0$, le pas.
Maintenir une table qui contient g_i le gradient de f_i pour $1 \leq i \leq N$ (à différents points)
for $k = 0, \ldots,$ **do**
  Tirer un indice i_k uniformément dans $\{1, \ldots, N\}$.
  Poser $g_{i_k}^{(k)} = \nabla f_i(x^{(k-1)})$ (gradient le plus récent)
  Poser $g_i^{(k)} = g_i^{(k-1)}$ pour $i \neq i_k$.
  $x^{(k)} = x^{(k-1)} - \alpha \left(g_{i_k}^{(k)} - g_{i_k}^{(k-1)} + \frac{1}{N}\sum_{i=1}^N g_i^{(k-1)}\right)$

A retenir

Il existe de nombreuses autres implémentations de gradient stochastique. Globalement le principe est le même : réduire la variance du gradient de la façon la plus efficace possible. Je donne simplement ici quelques titres de méthodes et renvoie le lecteur avide de connaissance à son moteur de recherche pour en savoir plus : SDCA, S2GD, MISO, Finito, SARAH, Katyusha...

Moyenner les itérées Un problème des méthodes de gradient stochastique est que la trajectoire finit par osciller autour d'un minimiseur sans jamais l'atteindre. Pour résoudre ce problème, on peut utiliser des techniques de moyennage : on évalue

$$\hat{w}_k = \sum_{k=1}^K \lambda_k w_k, \tag{3.17}$$

où $\lambda_k \geq 0$ et $\sum_{k=1}^K \lambda_k = 1$. Des choix typiques de la suite λ_k sont :

— $\lambda_K = 1$ et $\lambda_k = 0$ pour $k \neq K$. On retombe alors dans le gradient stochastique usuel.

— $\lambda_k = \frac{1}{K}$. On effectue alors une moyenne non pondérée des itérées. En pratique, cette approche est assez inefficace : les premières itérations sont loin du minimiseur et on n'a pas forcément envie de les moyenner.
— $\lambda_k = c_K \cdot k^\beta$, où c_K est une constante choisir pour que la somme des λ_k soit égale à 1 et où $\beta \geq 0$ est un exposant. Plus β est grand, plus l'importance relative des dernières itérées va être importante. Ce choix pour un paramètre β optimisé manuellement s'avère souvent fructueux en pratique.

A retenir

Moyenner les itérées a un coût numérique essentiellement nul et permet souvent de nettement améliorer la convergence. Je vous recommande donc d'effectuer des moyennes pour différentes suites (λ_k) et de choisir le meilleur résultat.

3.2.2 Recherche de pas

La recherche de pas (backtracking) est nécessaire dans de nombreux problèmes d'optimisation déterministes. L'intégration d'une recherche de pas pour le gradient stochastique n'est pas complètement triviale et des méthodes ont été proposées récemment dans [5]. La difficulté est de ne pas effectuer la recherche de pas sur la fonction f complète, car elle est coûteuse à évaluer, mais plutôt sur l'unique fonction f_{i_k} activée à l'itération k. L'idée est de trouver un pas α_k satisfaisant l'inéagilté de Armijo suivante

$$f_{i_k}(w_k - \alpha_k \nabla f_{i_k}(w_k)) \leq f_{i_k}(w_k) - c\alpha_k \|\nabla f_{i_k}(w_k)\|_2^2, \tag{3.18}$$

où $c > 0$ est un paramètre à fixer au départ (e.g. $c = 0.1$).

Sous certaines conditions techniques, on peut montrer que la descente de gradient stochastique converge sous la condition (3.18).

Je ne souhaite pas aller plus loin dans l'analyse de cette technique par manque de recul. Je souhaite cependant sensibiliser le lecteur à l'existence de cette possibilité et le renvoie à l'article [5] pour plus de détails techniques.

3.2.3 Préconditionnement diagonal

Tout comme en optimisation traditionnelle, il peut être bon (voire indispensable) de préconditionner le problème, i.e. de changer la métrique dans laquelle les gradients sont évalués. Dans le cas le plus simple, ceci donne lieu à des itérations du type :

$$w_{k+1} = w_k - M_k \nabla f_{i_k}(w_k),$$

où M_k est une matrice définie positive. Idéalement on aimerait qu'elle s'approche de l'inverse de la hessienne de f. Cependant, le calcul d'une inverse peut être bien trop gourmand en temps de calcul. On se contente donc de choisir les matrices M_k dans l'ensemble des matrices diagonales à coefficients positifs.

Ceci permet notamment de prendre en compte le fait qu'on optimise des poids qui ont des interprétations complétement différentes (e.g. biais et gains dans un réseau de

neurones). On verra que c'est une composante essentielle, probablement responsable du succès de l'algorithme ADAM. Comment choisir les coefficients A_i de $M_k = \mathrm{diag}(A_i)$? Deux algorithmes ont actuellement un succès important.

Exponential moving average Lors de l'entraînement de modèles d'apprentissage automatique, en particulier avec des méthodes de gradient stochastique, il est souvent utile de lisser les variables (poids, gradients,...) pour réduire leur variance. Une manière courante de le faire est d'utiliser la *moyenne mobile exponentielle* (ou *Exponential Moving Average*, EMA). Cette technique est utilisée dans des algorithmes comme **ADAM** pour stabiliser et accélérer la convergence. Soit (g_k) une suite de vecteurs.

On introduit une suite (m_k) définie par la récurrence suivante :

$$m_k = \beta m_{k-1} + (1-\beta) g_k, \tag{3.19}$$

où $\beta \in [0, 1[$ est un paramètre de lissage.

En développant la récurrence, on peut exprimer m_k sous une forme explicite :

$$m_k = \sum_{j=0}^{k} (1-\beta)\beta^{k-j} g_j. \tag{3.20}$$

Cette équation montre que m_k est une moyenne pondérée des gradients précédents, où les poids décroissent exponentiellement avec un facteur β.

Pour des petites valeurs de k, la suite m_k est biaisée, car les coefficients de pondération ne forment pas une somme égale à 1. En effet, on a :

$$\sum_{j=0}^{k} (1-\beta)\beta^{k-j} = 1 - \beta^{k+1} \neq 1. \tag{3.21}$$

Pour obtenir une combinaison convexe des g_k, il est nécessaire de normaliser m_k. On introduit alors une version corrigée $\hat{m}_k$:

$$\hat{m}_k = \frac{m_k}{1-\beta^{k+1}}. \tag{3.22}$$

Cette normalisation garantit que :

$$\hat{m}_k = \sum_{j=0}^{k} w_j g_j, \quad \text{avec} \quad w_j = \frac{(1-\beta)\beta^{k-j}}{1-\beta^{k+1}}, \tag{3.23}$$

et on vérifie que $\sum_{j=0}^{k} w_j = 1$. Ainsi, $\hat{m}_k$ est bien une combinaison convexe des g_k.

La moyenne mobile exponentielle présente plusieurs avantages :

— Elle n'introduit pas de surcoût de mémoire : il n'est pas nécessaire de stocker toute la suite des vecteurs g_k. Ils sont sommés au fur et à mesure qu'ils apparaissent.

— Elle permet de lisser les variations brusques dans les variables g_k, ce qui stabilise l'entraînement.

— Les vecteurs g_k récents ont plus de poids, ce qui permet d'adapter rapidement les mises à jour du modèle.

— Avec la normalisation, elle conserve une interprétation en termes de combinaison convexe.

RMSProp RMSProp est est l'acronyme de Root Mean Square Propagation algorithm. L'idée est de maintenir une moyenne de l'amplitude des composantes du gradient. Ceci est rendu possible par l'utilisation de l'algorithme EMA. On considère la suite

$$\forall j \in \{1, \dots, d\}, v_k = \beta v_{k-1} + (1-\beta) g_k^2$$

où g_k est le gradient stochastique, $\beta \in [0, 1[$ et g_k^2 représente le vecteur g_k dont on a élevé au carré toutes les composantes. On peut aussi définir $\hat{v}_k = v_k/(1-\beta^k)$. On peut alors considérer des itérations du type :

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{v_k + \mu}} g_k,$$

où la constante $\mu > 0$ est un paramètre de régularisation. Dans le cas où $\beta = 1$ et $\mu = 0$, on voit que l'itération devient :

$$w_k = w_k - \alpha \frac{g_k}{|g_k|}.$$

Ainsi, on normalise simplement l'amplitude de chaque composante du gradient. L'algorithme *devient invariant à des changements de métriques diagonales* ! En un sens, l'idée peut-être comprise comme une normalisation de l'influence des différentes composantes du gradient. Les garanties théoriques de cette approche sont encore mal comprises. Il serait cependant dommage de ne pas l'utiliser étant donné son succès empirique.

Adagrad Adagrad est une variante un peu plus simple de RMSprop que vous risquez de rencontrer. Je ne la décris pas ici, mais souhaitais simplement la mentionner étant donné son succès.

A retenir

Il est fréquent que les poids en apprentissage aient des interprétations complètement différentes et ne puissent pas être optimisées de la même façon. Le préconditionnement diagonal offre une solution simple pour prendre en compte ce phénomène. Je vous recommande de regarder les illustrations suivantes, en espérant qu'elles soient pérennes.

3.2.4 Inertie / Moyennage des gradients

Vous avez dû voir dans le cours d'optimisation convexe, qu'il était possible d'accélérer les méthodes de gradient significativement en utilisant de l'inertie : les gradients aux itérations précédentes sont utilisées. On peut faire une analogie avec une boule qui descent sur une montagne : elle ne suis pas la pente maximale (comme le fait la descente de gradient), mais son mouvement est aussi régi par l'inertie de la boule. Dans un cadre stochastique, les méthodes inertielles prennent la forme suivante :

$$w_{k+1} = w_k - \alpha \nabla f_{i_k}(w_k) + \beta(w_k - w_{k-1})$$

où $\alpha > 0$ est le pas de la descente de gradient et $\beta > 0$ est un paramètre d'inertie. Quitte à réorganiser les termes dans la suite, on voit que

$$w_{k+1} = w_k - \alpha \sum_{j=0}^{k-1} \beta^{k-j} \nabla f_{i_k}(w_j),$$

i.e. le pas de descente est défini comme une combinaison linéaire des gradients aux itérations précédentes avec des poids décroissants.

ADAM L'algorithme ADAM (adaptive moment estimation) est probablement encore l'algorithme d'optimisation stochastique le plus populaire début 2021. Il reprend l'essentiel des idées évoquées précédemment. Il est décrit dans l'Algorithme 4.

Il dépend de plusieurs paramètres :

— α : le pas de la descente. Comme les composantes du gradient sont normalisées, il n'y a plus à se soucier de la constante de Lipschitz. Il est typiquement fixé à 0.001 et varie dans l'intervalle $[10^{-5}, 0.5]$.
— β_1 : il est typiquement fixé à 0.9 et contrôle l'inertie du premier ordre.
— β_2 : typiquement fixé à 0.999 et contrôle l'inertie du second ordre.
— ϵ : petit paramètre (typiquement 10^{-8}) pour éviter les divisions par 0. La documentation de TensorFlow indique qu'empiriquement des valeurs plus grandes peuvent être préférables.

Algorithm 4: La méthode ADAM.

Input:
$w^{(0)}$: un point de départ. $\alpha > 0$, le pas.
Poser $m_0 = 0$ (moment du premier ordre)
Poser $v_0 = 0$ (moment du second ordre)
for $k = 0, \ldots, K$ **do**
  $g_k = \nabla f(w_{k-1})$ (gradient courant)
  $m_k = \beta_1 m_{k-1} + (1 - \beta_1) g_k$ (mise à jour du premier moment)
  $v_k = \beta_2 v_{k-1} + (1 - \beta_2) g_k^2$ (mise à jour moment d'ordre 2)
  $\hat{m}_k = m_k / (1 - \beta_1^k)$ (estimation non biaisée du moment d'ordre 1)
  $\hat{v}_k = v_k / (1 - \beta_2^k)$ (estimation non biaisée du moment d'ordre 2)
  $w_k = w_{k-1} - \alpha \hat{m}_k / \sqrt{(\hat{v}_k + \epsilon)}$
Renvoyer w_K.

Les avantages de cette approche sont nombreux :

— Très facile à implémenter.
— Efficace empiriquement.
— Peu coûteux numériquement.
— Peu de besoins en mémoire.
— Invariance aux changements de métriques diagonales.
— Les paramètres ont une interprétation physique et leur ajustement manuel se révèle assez facile en pratique (avec un peu d'expérience tout de même).

A retenir

L'algorithme ADAM est aujourd'hui un des plus populaire pour l'entraînement de réseaux de neurones. On lui reproche quelques-fois ses capacités limitées de généralisation (il fait de l'over-fitting). Ses garanties théoriques sont encore maigres. Cependant, c'est une méthode de choix pour des problèmes difficiles qui a fait ses preuves dans de nombreux contextes.

3.2.5 Contraintes et régularisation

Jusqu'à présent, nous avons considéré l'optimisation d'une fonction $f : \mathbb{R}^d \to \mathbb{R}$ différentiable sans contrainte. Dans certaines situations, il peut être nécessaire contraindre ou de pénaliser les poids w et de résoudre des problèmes de type :

$$\min_{w \in \mathcal{W}} f(w), \tag{3.24}$$

où $\mathcal{W}$ est une contrainte de régularisation. Autoriser la fonction à valoir $+\infty$ permet de couvrir le cas de contraintes (e.g. positivité) et de termes non différentiables (e.g. normes ℓ^1).

Comment intégrer le régulariseur ou les contraintes dans le cadre stochastique ? On pourrait penser qu'un simple ajout de projecteur ou d'opérateur proximal (une descente de gradient stochastique proximale) pourrait suffire. Malheureusement, ce n'est pas le cas, même dans le cas convexe.

Pour traiter ces contraintes, une possibilité est d'utiliser un algorithme appelé extra-gradient, utilisé pour la résolution d'inégalités variationnelles (une généralisation des problèmes d'optimisation). Je ne souhaite pas rentrer dans cette théorie ici, mais simplement décrire une version stochastique pour la résolution du problème (3.24). Cette variante est appelée extra-ADAM, en référence à l'algorithme Adam et à la méthode d'extra-gradient. Elle a été proposée dans [17]. Elle est donnée dans l'Algorithme 5.

Algorithm 5: La méthode extra-ADAM pour le cas contraint.

Input:

$w^{(0)}$: un point de départ. $\alpha > 0$, β_1 et β_2 les paramètres d'Adam

Poser $m_0 = 0$ (moment du premier ordre)

Poser $v_0 = 0$ (moment du second ordre)

for $k = 0, \dots, K$ **do**

$g_k = \nabla_{i_{k-1/2}} f(w_k)$

$m_{k-1/2} = \beta_1 m_{k-1} + (1-\beta_1) g_k$

$v_{k-1/2} = \beta_2 v_{k-1} + (1-\beta_2) g_k^2$

$\hat{m}_{k-1/2} = m_{k-1/2}/(1-\beta_1^{2k-1})$

$\hat{v}_{k-1/2} = v_{k-1/2}/(1-\beta_2^{2k-1})$

$w_{k+1/2} = \Pi_{\mathcal{W}}\left(w_k - \alpha \frac{\hat{m}_{k-1/2}}{\sqrt{\hat{v}_{k-1/2}+\epsilon}}\right)$

$g_{k+1/2} = \nabla_{i_{k+1/2}} f(w_{k+1/2})$

$m_k = \beta_1 m_{k-1/2} + (1-\beta_1) g_{k+1/2}$

$v_k = \beta_2 v_{k-1/2} + (1-\beta_2) g_{k+1/2}^2$

$\hat{m}_k = m_k/(1-\beta_1^{2k})$

$\hat{v}_k = v_k/(1-\beta_2^{2k})$

$w_{k+1} = \Pi_{\mathcal{W}}\left(w_k - \alpha \frac{\hat{m}_k}{\sqrt{\hat{v}_k+\epsilon}}\right)$

Renvoyer w_K.

3.3 Extensions

Ce cours a présenté quelques grands principes de l'optimisation stochastique pour l'apprentissage supervisé. Ces techniques d'optimisation stochastiques ont de nombreuses autres applications qui pourraient rentrer dans le cadre de ce cours. Il existe aussi d'autres approches importantes pour l'apprentissage. Ainsi, ce cours pourrait être étendu avec les aspects suivants :

— Descentes par coordonnées (méthode duale). Dualité, systèmes linéaires immenses.

— Modèles de diffusion (diffusion de Langevin, recuit simulé).

Dans l'immédiat cependant, je préfère laisser ces notes telles quelles, mais le cours pourrait évoluer vers ces approches à terme.

Bibliographie

[1] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[2] Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. Introduction to statistical learning theory. In *Summer School on Machine Learning*, pages 169–207. Springer, 2003.

[3] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20 :161–168, 2007.

[4] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2) :223–311, 2018.

[5] Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless stochastic gradient : Interpolation, line-search, and convergence rates. In *Advances in Neural Information Processing Systems*, pages 3732–3745, 2019.

[6] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning : From theory to algorithms*. Cambridge university press, 2014.

[7] Allan Pinkus. *N-widths in Approximation Theory*, volume 7. Springer Science & Business Media, 2012.

[8] Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

[9] Philipp Grohs, Dmytro Perekrestenko, Dennis Elbrächter, and Helmut Bölcskei. Deep neural network approximation theory. *arXiv preprint arXiv :1901.02220*, 1, 2019.

[10] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[11] Roman Vershynin. *High-dimensional probability : An introduction with applications in data science*, volume 47. Cambridge university press, 2018.

[12] Mikhail Belkin. Fit without fear : remarkable mathematical phenomena of deep learning through the prism of interpolation. *Acta Numerica*, 30 :203–248, 2021.

[13] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning : data mining, inference, and prediction*, volume 2. Springer, 2009.

[14] Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. Gradient descent only converges to minimizers. In *Conference on learning theory*, pages 1246–1257, 2016.

[15] Lénaïc Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. *Advances in Neural Information Processing Systems*, 31 :3036–3046, 2018.

[16] Raphaël Barboni, Gabriel Peyré, and François-Xavier Vialard. Global convergence of resnets : From finite to infinite width using linear parameterization. *arXiv preprint arXiv :2112.05531*, 2021.

[17] Gauthier Gidel, Hugo Berard, Gaëtan Vignoud, Pascal Vincent, and Simon Lacoste-Julien. A variational inequality perspective on generative adversarial networks. *arXiv preprint arXiv :1802.10551*, 2018.