“Algorithmes stochastiques” - Examen partiel

Université de Paris - M2 MMA

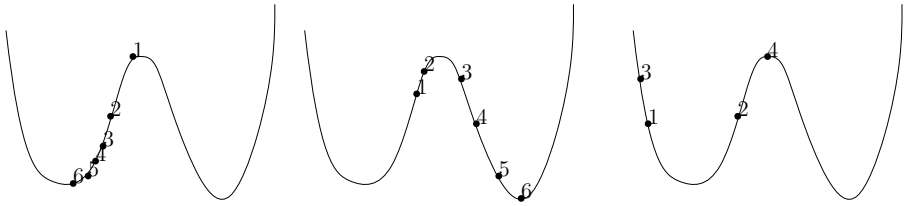
8 novembre 2021 (durée : trois heures)

- Ce sujet est composé d’une partie “théorique” (à composer sur papier) et d’une partie “pratique” (à composer sur ordinateur).
- Il faut rendre votre copie “théorique” avant de passer à la “pratique”. Cependant, vous disposez déjà du sujet “pratique” et pouvez y réfléchir à l’écrit.
- Les notes manuscrites sont autorisées pendant tout l’examen. Pendant la partie sur ordinateur, vous pouvez consulter la documentation Python et les corrigés des TP mais il est évidemment interdit de recopier du code directement depuis des sources externes, et toute forme de communication est interdite.
- Pour la partie “théorique” :
  - Sauf indication contraire, toute réponse demande une argumentation mathématique précise.
  - Je vous propose les parcours suivants :
    1. Tout le monde fait la première question de “type cours”.
    2. Vous choisissez entre le sujet 1 (plus théorique) et le sujet 2 (qui s’appuie sur un article).
    3. Si vous êtes à court d’inspiration pendant l’examen, vous pouvez regarder la deuxième question de “type cours”.

    Il est bien sûr possible d’alterner entre les deux sujets mais je vous le déconseille.
- Pour la partie “pratique” :
  - Il s’agit d’implémenter des algorithmes décrits dans un article joint. On ne fera pas aujourd’hui d’expérimentations très intéressantes, le but est “simplement” de comprendre ce que font les algorithmes et de les coder correctement en Python.
  - Un code lisible, même s’il renvoie des erreurs, sera valorisé.
- N’hésitez pas à poser des questions.
- Bon courage !

Questions de type cours

1. Sur les dessins ci-dessous, on a représenté le graphe d'une fonction de coût (définie sur un certain intervalle de la droite réelle) dont on chercher à trouver le minimum (et le minimiseur) en implémentant un algorithme de type "descente de gradient". Les points correspondent aux valeurs successives (du coût) obtenues par l'algorithme, dans l'ordre indiqué par les numéros. Le point numéro 1 correspond au coût pour la valeur initiale du paramètre que l'on a choisie au démarrage de l'algorithme.



   (a) En justifiant rapidement, dire lequel des dessins correspond à :
      i. Un choix trop grand pour le pas de temps.
      ii. Une situation impossible.

   (b) Pour le dessin restant, que se passe-t-il ? Est-ce en contradiction avec le résultat de convergence énoncé et démontré en cours ?

   (c) La situation "impossible" serait-elle toujours impossible si on utilisait une descente de gradient *stochastique* ?

   (d) La dernière situation (de la question (b)) serait-elle améliorée si on utilisait une descente de gradient *stochastique* ?

   (e) Imaginons que la fonction de coût soit donnée par $F(x) = x^3 - 10(x-1)^2$, étudiée sur l'intervalle $[0, 8]$. Quelle valeur numérique du pas de temps choisiriez-vous et pourquoi ?

2. On rappelle que les *lois exponentielles* forment une famille à un paramètre de lois de probabilité définies ainsi : pour tout réel λ strictement positif, on note $\mathcal{E}_\lambda$ la densité suivante (sur $[0, +\infty]$) :

$$\mathcal{E}_\lambda(x) := \lambda \exp(-\lambda x)$$

   En cas de besoin, on pourra utiliser sans justification le fait que la moyenne (resp. la variance) de $\mathcal{E}_\lambda$ est égale à $\frac{1}{\lambda}$ (resp. $\frac{1}{\lambda^2}$).

   (a) Soit $n \geq 1$ et soit $(x_1, \ldots, x_n)$ un échantillon d'une loi exponentielle de paramètre λ inconnu. Montrer que l'estimateur $\hat{\lambda}$ défini par :

$$\frac{1}{\hat{\lambda}} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

   est un point critique de la log-vraisemblance. (Il s'agit en fait de l'estimateur du maximum de vraisemblance).

   (b) On considère maintenant la famille paramétrique formée de tous les mélanges possibles de K lois exponentielles. Donner (sans justifier) la forme générale d'un élément de cette famille. À K fixé, combien y a-t-il de paramètres dans cette famille ?

   (c) Proposer (en détaillant votre réponse) un algorithme pour calculer numériquement l'estimateur de vraisemblance d'un modèle de mélange de lois exponentielles.

Théorie 1 : étude de l'algorithme EM près d'une situation dégénérée

D'après Biernacki, C. & Chrétien, S. (2002) *Degeneracy in the maximum likelihood estimation of univariate Gaussian mixtures with EM* Statistics & Probability Letters, 61(4), 373-382,

Pour $\mu \in (-\infty, +\infty)$ et $\sigma > 0$ on notera $g_{\mu,\sigma}$ la densité gaussienne donnée par

$$g_{\mu,\sigma}(x) := \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

Dans toute la suite, K est un entier ≥ 2 fixé.

1. Rappeler avec précision l'expression de la densité d'un mélange de K gaussiennes (appelé simplement "GMM" ici).

   *Dans la suite on notera θ le vecteur des paramètres d'un GMM, en incluant les poids $p_1, \ldots, p_K$. On dit qu'un GMM **dégénère** si l'une des variances tend vers 0 (par exemple au cours d'un algorithme).*

2. Soit $n \geq 2$ et soit $\vec{x} = (x_1, \ldots, x_n)$ un échantillon obtenu à partir d'un GMM de paramètres inconnus. Donner l'expression de la vraisemblance (likelihood) $\theta \mapsto L(\theta, \vec{x})$.

3. Si la moyenne μ_k de l'une des composantes coïncide avec la valeur x_i de l'une des observations (c'est à dire : si $\mu_k = x_i$ pour un certain $k \in \{1, \ldots, K\}$ et un certain $i \in \{1, \ldots, n\}$) et si l'écart-type σ_k correspondant tend vers 0 (les autres paramètres étant fixés), montrer que la vraisemblance $L(\theta, \vec{x})$ diverge comme

   $$L(\theta, \vec{x}) \sim \frac{M}{\sigma_k}$$

   où M est une constante. En déduire que le problème de maximisation de la vraisemblance est mal posé.

4. Quelles modifications pourriez-vous apporter au problème d'optimisation considéré afin de contourner cette difficulté ?

5. Pourquoi cette difficulté n'est-elle pas présente dans le cas d'*une seule* loi gaussienne ($K = 1$) ?

*Le but de ce sujet est d'étudier le comportement de l'algorithme EM au voisinage d'une solution dégénérée. En particulier, on montrera que (sous certaines hypothèses) la solution dégénérée est (malheureusement) attractive, avec vitesse de convergence exponentielle. Pour simplifier, on supposera que c'est toujours la **première** composante du mélange qui dégénère en étant très fortement piquée autour de la **première** observation. Plus précisément, pour $D > 0$ (on pensera à D comme étant "très grand") on dira que le vecteur de paramètres θ est D-dégénéré lorsque :*

$$p_1 g_{\mu_1,\sigma_1}(x_1) \geq D.$$

Cela revient à dire que la première composante du mélange a une très grande densité en la première observation (notons que le poids p_1 est inclus dans l'expression). Toujours pour simplifier, on va supposer qu'il ne se passe "rien" avec les autres observations et les autres composantes du mélange, à savoir que toutes les autres densités en chaque observation sont de taille "normale". Plus précisément, pour $C > 0$ (on pensera à C comme étant "d'ordre 1") on dira que le vecteur de paramètres θ est C-contrôlé lorsque :

$$\max_{i=2\ldots n} p_1 g_{\mu_1,\sigma_1}(x_i) \leq C \text{ et } \max_{k=2\ldots K} \max_{i=1\ldots n} p_k g_{\mu_k,\sigma_k}(x_i) \leq C.$$

6. Représenter grossièrement sur un dessin une situation dégénérée et contrôlée.

Soit θ^t les paramètres obtenus à l'étape t de l'algorithme (on note $p_1^t, \dots, p_K^t$ les poids, $\mu_1^t, \dots, \mu_K^t$ les moyennes et $\sigma_1^t, \dots, \sigma_K^t$ les écarts-types). On suppose que le mélange est déjà un peu dégénéré à l'étape t et on va chercher à décrire θ^{t+1}. ***Plus précisément, dans la suite on fixe D et C et on suppose que θ^t est à la fois D-dégénéré et C-contrôlé.***

7. (a) Rappeler en quoi consiste la première étape de l'algorithme EM (la formule et l'explication de ce qu'on fait).

   (b) Établir que les paramètres γ_{ik}^t calculés par l'algorithme possèdent les développements suivants :

$$\gamma_{11}^t = 1 - O\left(\frac{nC}{D}\right),\ \gamma_{i1}^t = O\left(\frac{C}{D}\right) \text{ pour } i = 2, \dots, n,\ \gamma_{1k}^t = O\left(\frac{C}{D}\right) \text{ pour } k = 2, \dots, K.$$

8. (a) Rappeler en quoi consiste la seconde étape de l'algorithme EM *pour un GMM* (les formules et l'explication de ce qu'on fait).
   Dans la suite on note $\|x\|_\infty$ la taille maximale des observations faites (le maximum de $|x_1|, \dots, |x_n|$).

   (b) Montrer que le paramètre p_1^{t+1} vérifie :

$$p_1^{t+1} \geq \frac{1 - O\left(\frac{nC}{D}\right)}{n}$$

   (c) Montrer que le paramètre μ_1^{t+1} vérifie :

$$\mu_1^{t+1} - x_1 = O\left(\frac{nC\|x\|_\infty}{D^2}\right).$$

   (d) Montrer que le paramètre σ_1^{t+1} vérifie :

$$(x_1 - \mu_1^{t+1})^2 \left(1 - O\left(\frac{nC}{D}\right)\right) \leq \left(\sigma_1^{t+1}\right)^2 \leq (x_1 - \mu_1^{t+1})^2 + \frac{4nC\|x\|_\infty}{D^2}.$$

   (e) Déduire des questions précédentes une minoration de la densité $p_1^{t+1} g_{\mu_1^{t+1}, \sigma_1^{t+1}}(x_1)$ en fonction de $D, n, C, \|x\|_\infty$.

   (f) ($\star$) Montrer (en admettant des propriétés si besoin) que si à une étape t de l'algorithme le rapport $\frac{D^t}{C^t}$ est plus grand qu'une certaine constante (dépendant uniquement du nombre et de la taille des observations), alors la quantité $p_1^t g_{\mu_1^t, \sigma_1^t}(x_1)$ diverge ensuite vers $+\infty$ exponentiellement vite.

   (g) ($\star\star$) Finir la preuve en donnant les arguments/calculs manquants à l'étape précédente (s'il y en avait !).

Théorie 2 : algorithme EM pour la détection de motifs en biologie

D'après :

— Lawrence, C. E., & Reilly, A. A. (1990). *An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences* Proteins, 7(1), 41–51. **Il s'agit de l'article principal.**

— Fan, X. & Yuan, Y. & Liu, J.S. (2010). *The EM Algorithm and the Rise of Computational Biology* Statistical Science, Statist. Sci. 25(4), 476-491. **Un extrait de cet article est joint. Sa lecture est optionnelle, elle peut (ou non) éclairer le premier article.**

Notation. Soit $\mathcal{A}$ un alphabet fini, c'est à dire un ensemble fini de d lettres (on supposera qu'il y a au moins deux lettres dans l'alphabet i.e. $d \geq 2$). Un mot sur l'alphabet $\mathcal{A}$ est une suite finie de lettres notée $\mathbf{Y} = (\mathbf{Y}_1, \ldots, \mathbf{Y}_L)$, où $L \geq 1$ est la longueur du mot, et chaque caractère $\mathbf{Y}_i$ ($1 \leq i \leq L$) est une lettre de l'alphabet $\mathcal{A}$. On notera $\mathcal{M}$ l'ensemble des mots (finis et possiblement vides) et $\mathcal{M}_L$ l'ensemble des mots de longueur L fixée ($L = 0$ correspond au mot vide).

1. Expliquer pourquoi l'ensemble des lois de probabilité sur l'alphabet $\mathcal{A}$ (c'est-à-dire la loi d'une seule lettre aléatoire parmi les d lettres possibles) forme une famille paramétrique, et en donner une représentation naturelle (indice : vecteur de somme 1).

2. Soit $L \geq 1$ fixé. On dira qu'une loi de probabilité $\mathbb{P}$ sur $\mathcal{M}_L$ est "de type iid" s'il existe une loi de probabilité μ sur *l'alphabet* $\mathcal{A}$ telle que :

$$\mathbb{P}\left(\mathbf{Y} = (\mathbf{Y}_1, \ldots, \mathbf{Y}_L)\right) = \prod_{i=1}^{L} \mu(\mathbf{Y}_i).$$

   Cela correspond bien sûr au fait que chaque caractère du mot est tiré indépendamment selon la loi μ.

   (a) D'après la question 1. les lois sur $\mathcal{A}$ forment une famille paramétrique, il en est donc de même pour les lois "de type iid" sur $\mathcal{M}_L$. Étant donné **un seul** mot $\mathbf{Y}$ de longueur L, que l'on suppose être la réalisation d'une loi "de type iid" de paramètre inconnu, donner l'expression de la vraisemblance comme fonction du paramètre et des caractères du mot.

   (b) Donner un estimateur naturel du vecteur des paramètres.

3. Soit $L \geq 1$ fixé. On dira qu'une loi de probabilité $\mathbb{P}$ sur $\mathcal{M}_L$ est "de type i, pas id" s'il existe des lois $\mu_1, \ldots, \mu_L$ de probabilité sur l'alphabet $\mathcal{A}$ telles que :

$$\mathbb{P}\left(\mathbf{Y} = (\mathbf{Y}_1, \ldots, \mathbf{Y}_L)\right) = \prod_{i=1}^{L} \mu_i(\mathbf{Y}_i).$$

   Cela correspond au fait que chaque caractère est tiré indépendamment, mais sa loi peut dépendre de sa position dans le mot.

   (a) Montrer qu'il s'agit encore d'une famille paramétrique, et en donner une représentation (indice : matrice de taille $L \times d$).

   (b) Est-ce que toutes les lois sur $\mathcal{M}_L$ sont de type "iid" ou "i, pas id" ?

4. On s'intéresse maintenant à des situations où un certain motif se cache au sein d'un mot. On supposera que la longueur w du motif est connue. Un motif de longueur w au sein d'un mot $\mathbf{Y}$ de longueur L est un sous-mot $\mathbf{M}$ de w lettres consécutives, de la forme $\mathbf{M} = (\mathbf{Y}_i, \mathbf{Y}_{i+1}, \ldots, \mathbf{Y}_{i+w-1})$ pour

un certain i entre 1 et $L - w + 1$. Si le motif $\mathbf{M}$ apparaît dans le mot $\mathbf{Y}$, on décomposera $\mathbf{Y}$ sous la forme $\mathbf{Y}^{(1)} \cdot \mathbf{M} \cdot \mathbf{Y}^{(2)}$, où $\cdot$ dénote la concaténation de deux mots. Par exemple si $\mathbf{Y}$ est le mot $(A, A, T, A, G, C, G, C, C, A, T, G, C, T, C)$ sur l'alphabet $\mathcal{A} = \{A, T, G, C\}$ à $d = 4$ lettres, et $\mathbf{M}$ est le motif (G, C, C, A) on a bien $\mathbf{Y} = (A, A, T, A, G, C) \cdot \mathbf{M} \cdot (T, G, C, T, C)$.

(a) Si le motif $\mathbf{M}$ est **connu**, décrire (rapidement) un algorithme simple (et déterministe) pour vérifier la présence de $\mathbf{M}$ au sein d'un mot $\mathbf{Y}$.

(b) On suppose maintenant que $\mathbf{M}$ n'est **pas connu**, mais que la présence d'un motif peut se détecter comme une anomalie dans la distribution des lettres. Plus précisément, on suppose que si on décompose un mot $\mathbf{Y}$ contenant le motif comme $\mathbf{Y} = \mathbf{Y}^{(1)} \cdot \mathbf{M} \cdot \mathbf{Y}^{(2)}$, alors $\mathbf{Y}^{(1)}$ et $\mathbf{Y}^{(2)}$ sont deux mots suivant une loi "de type iid" associée à une certaine mesure μ_0 sur $\mathcal{A}$, tandis que le motif $\mathbf{M}$ suit une loi "de type i, pas id" sur $\mathcal{M}_w$ associée à des lois $(\mu_1, \ldots, \mu_w)$ sur $\mathcal{A}$.

i. Si la position du motif $\mathbf{M}$ au sein du mot $\mathbf{Y}$ est connue, expliquer comment estimer les paramètres de la loi μ_0 à partir des lettres de $\mathbf{Y}$.

ii. Peut-on estimer les paramètres de la loi du motif à partir du seul mot $\mathbf{Y}$?

iii. Et si on dispose d'un échantillon de n mots, contenant chacun un motif, *la position du motif dans chaque mot étant connue* (mais pouvant varier d'un mot à l'autre), peut-on estimer tous les paramètres ?

(c) On suppose maintenant qu'on ne connaît ni la loi μ_0, ni la loi du motif, ni la position du motif. Pour simplifier on pourra supposer que les mots $\mathbf{Y}$ ont tous la même longueur L avec $L \geq w$. Décrire la situation :

i. Comme un modèle de mélange (on précisera les éléments de la famille, à quoi correspondent les poids etc.)

ii. Comme un couple $(\mathbf{Z}, \mathbf{Y})$ où $\mathbf{Z}$ est une variable latente/cachée que vous définirez et $\mathbf{Y}$ est un mot aléatoire (on décrira la loi jointe de $(\mathbf{Z}, \mathbf{Y})$).

(d) **Proposer un algorithme pour estimer les paramètres du modèle.**

5. **Quel est le rapport avec l'article ?** En particulier :
   (a) À quel aspect du modèle fait référence la phrase indiquée par #1 ?
   (b) À quel aspect du modèle fait référence l'expression "mononucleotide/monoresidue model" (#2, #3 entre autres) ?
   (c) À quel aspect du modèle fait référence la phrase indiquée par #4 ?
6. **L'appendice A a très malencontreusement disparu. Reconstituez les calculs qui s'y trouvent, en relation avec la phrase #5.**
7. Il est question (#6) de motifs "palindromiques". De quoi s'agit-t-il à votre avis ? Comment pourrait-on changer notre modèle pour prendre en compte cette hypothèse biologique ?
8. Même question pour la phrase #7.

PROTEINS: Structure, Function, and Genetics 7:41–51 (1990)

An Expectation Maximization (EM) Algorithm for the Identification and Characterization of Common Sites in Unaligned Biopolymer Sequences

Charles E. Lawrence and Andrew A. Reilly
Biometrics Laboratory, Wadsworth Center for Laboratories and Research, New York State Department of Health, Albany, New York 12201

ABSTRACT **Statistical methodology for the identification and characterization of protein binding sites in a set of unaligned DNA fragments is presented. Each sequence must contain at least one common site. No alignment of the sites is required. Instead, the uncertainty in the location of the sites is handled by employing the missing information principle to develop an "expectation maximization" (EM) algorithm. This approach allows for the simultaneous identification of the sites and characterization of the binding motifs. The reliability of the algorithm increases with the number of fragments, but the computations increase only linearly. The method is illustrated with an example, using known cyclic adenosine monophosphate receptor protein (CRP) binding sites. The final motif is utilized in a search for undiscovered CRP binding sites.**

Key words: DNA binding proteins, maximum likelihood, CRP, finite mixtures, transcription regulation

INTRODUCTION

The identification of common sites in multiple sequences is frequently encountered in the analysis of biopolymer sequence data. Examples are protein-binding sites in DNA sequences, T-cell binding sites, and antigenic epitopes of protein sequences. Although the methods we present are applicable to a broad class of such problems, we focus here on DNA protein-binding sites. Such sites have traditionally been identified by isolating cis-acting mutations that affect expression and determination of corresponding changes in the DNA of the mutant phenotypes.[1] More recent techniques include affinity purification of the DNA-binding regions and "footprinting" techniques.[2] These methods are time-consuming and provide partial information about the binding sites; the final determination of the sites usually requires the comparison of many examples. One aim of the methods proposed here is to reduce the required experimental work to the identification of restriction fragments that contain binding sites.

The diversity that arises in such fragments is illustrated by *Escherichia coli* promoter sites. Consensus sequences at −35 and −10 are, respectively, "TTGACA" and "TATAAT," but none of the positions are absolutely conserved. The most conserved bases at the −10 position are "TAxxxT," but only about 65% of all promoters match even this criterion. Deuschle et al.[3] have shown that promoters of identical strength exhibit different structures through optimization of different elements of the promoter sequence. This leads to a diversity of functioning sequences, all of which depart substantially from the consensus. Consensus descriptions are therefore likely to have important limitations. Methods that focus on matching identical substrings of "words" share in some of these limitations.[4,5] The identification of sites common to several sequences is consequently linked with the model employed to describe the diversity.

A better description of sites is a stochastic model of residue frequencies. This model assigns probabilities to each of the four bases at each position in the site. An information measure based on a matrix of these probabilities has been shown to provide a useful indication of how constrained the choice of bases is at each site.[6,7] The measure is highly correlated with binding affinities when binding sites are known.[8]

Recently, Stormo and Hartzell,[9] proposed an algorithm to identify protein binding sites in unaligned DNA fragments that uses this measure. They focus exclusively on a mononucleotide/monoresidue model: Each position in the site has residue probabilities independent of any other position. However, the structural features of the proteins involved in site selection transcend the features encompassed in a monoresidue model. The most important limitation of monoresidue models stems from their assumption of no correlative effects across multiple residues. For

Received July 10, 1989, revision accepted September 25, 1989.

Address reprint requests to Dr. Charles E. Lawrence, Biometrics Laboratory, Wadsworth Center for Laboratories and Research, New York State Department of Health, Albany, NY 12201.

© 1990 WILEY-LISS, INC.

example, in dimeric prokaryotic DNA binding proteins, residues from the helix-coil-helix motifs make contact with bases in two major groove openings. The symmetry of the dimer leads to a strong correlation in the base frequencies at equivalent yet nonadjacent positions in the major grooves, resulting in a palindromic pattern.

The method presented here is designed to capture and characterize such structural features by employing a set of models derived from the proposed motifs. These models and the associated statistical test procedures allow for the characterization of the binding motif and improved identification of the binding sites. Additionally, a mechanism for determining the length of the site is presented. To consider these various alternative models, we employ a generalization of the information measure presented by Storma and Hartzell, the log likelihood.

The log likelihood is at the heart of a very general procedure for data analysis, maximum likelihood estimation,[10] and the method presented here. Unaligned sequences contain no explicit information on the location of the sites of interest. In multiple sequence comparison problems, this leads to a high degree of positional uncertainty. We employ the "missing information principle" to develop an expectation maximization (EM) algorithm that overcomes this informational deficit. The EM algorithm is described by Dempster, Laird, and Rubin[11] and in a recent text by Little and Rubin.[12] For our purposes, it suffices that it has been shown to be applicable to a broad range of problems, including many problems not normally considered to arise from missing or incomplete data. The applications that are the closest to ours are latent class models[13] and estimation in finite mixture models.[14] We have found that the formulation of this problem as a special class of finite mixture models leads to a number of useful statistical and informational insights, to be presented elsewhere. The algorithm generates estimates of the probabilities that the sites are located in each possible position in each sequence and thereby predicts the most likely binding sites simultaneously with maximum likelihood estimates of the model parameters.

#3

We begin with the mononucleotide model and then show how to incorporate specific motif characteristics through the use of models appropriate to each. The procedure also provides a means for selecting the most data-consistent model from the set of models specified. We apply the method using DNA fragments that are known to contain sites that bind to cyclic adenosine monophosphate receptor protein (CRP)). Using the final selected model, we scan a large database for undiscovered binding sites. This procedure locates four new sites and gives a frame of reference for judging the importance of secondary sites.

MATERIALS AND METHODS

The Problem

To facilitate the description of the algorithm, we begin by posing the problem using CRP binding sites as an example. This protein and its binding sites have been extensively studied, and much is known about its recognition sites.[15,16] CRP binds to several sites on the *E. coli* genome, where it can function either to enhance or to repress gene expression. Figure 1 shows several of these sites, each 22 bases long, from 18 sequences, and a tabular histogram of the occurrence of each base at each position.

Figure 2 shows the data to be analyzed. In these data, the sequences of 105 bases have been selected to position the sites randomly within the sequences.[9] The second row for each sequence marks the location of the start of each site with a 1. Solely to enhance the reader's ability to visualize the sites within the sequences, we have capitalized the bases within each site.

To state the problem succinctly, we seek to reproduce Figure 1 and second rows of Figure 2, using only the sequence information, i.e., the top rows, of Figure 2. Suppose it is known that a protein binds to at least one site of length J, 22 positions in this example, in each of these fragments, but that the position of the binding site(s) in each fragment is unknown. If the fragments are L, here 105, bases long, then there are (L − J), here 83, positions outside the site. Since the location of the site in any of the sequence is unknown, the site could be in any of (L − J + 1), 84, positions in each sequence. There are consequently 84^{18} combinations of segments of 22 bases, from which the correct 18 must be chosen. If we knew where the sites where, i.e., if complete data were available, then the base probabilities, $\rho_{b,j}$, $j = 1,2 \ldots J$; b = A,C,G,T for the positions within the site, and the base composition for all positions outside the site, $\rho_{b,O}$, can be estimated from the collection of marked subsequences (see Fig. 1). Our problem is that the site location information is missing. We are challenged to find the site locations and the base probabilities, $\rho_{b,j}$, using only the sequence data $S_1 \ldots S_N$.

#4

The Algorithm

The EM algorithm simplifies the analysis of problems with missing information by iteratively solving a sequence of problems in which expected information is substituted for missing information. This expected information is used at each step to solve the more straightforward problem associated with having complete information, by maximum likelihood. Thus the first step is to find the form of the solution as if one had complete information.

In our case, we are missing positional information. Thus we begin by formulating the problem as if we had the missing positional information. Given the

Footprint Sites

Col E1 site 2	T T T T T T G A T C G T T T T C A C A A A A
Col E1 site 1	T T T T T G T G G C A T C G G G C G A G A A T
ara site 2	T T A T T T G C A C G G C G T C A C A C T T
ara site 1	A A A A G T G T C T A T A A T C A C G G C A
Bgl R mut1	A A C T G T G A G C A T G G T C A T A T T T
crp	G T A T G C A A A A G G A C G T C A C A T T A
cya	A G G T G T T A A A T T G A T C A C G T T T
deo P2 site 2	T T A T T T G A A C C A G A T C G C A T T A
deo P2 site 1	A A T T G T G A T G T G T A T C G A A G T G
gal	T A A T T T A T T C C A T G T C A C A C T T
ilv B	A A A C G T G A T C A A C C C C T C A A T T
lac site 2	T A A T G T G A G T T A G C T C A C T C A T
lac site 1	G A A T T G T G A G C G G A T A A C A A T T
mal E	T T C T G T A A C A G A G A T C A C A C A A
mal K	T T C T G T G A A C T A A A C C G A G G T C
mal T	A A T T G T G A C A C A G T G C A A A T T C
omp A	A T G C C T G A C G G A G T T C A C A C T T
tna A	G A T T G T G A T T C G A T T C A C A T T T
uxu AB	T G T T G T G A T G T G G T T A A C C C A A
Pbr P4	C G G T G T G A A A T A C C G C A C A G A T
cat	A A A A T G A G A C G T T G A T C G G C A C

A) Base frequencies in footprint sites

A	8	10	9	2	0	0	4	15	8	5	3	10	3	7	1	2	15	4	14	4	7	6
C	1	0	3	2	1	1	0	1	5	8	5	1	4	3	2	18	1	15	1	7	1	3
G	3	3	3	0	14	2	15	3	2	5	6	5	10	6	3	0	4	1	5	4	0	1
T	9	8	6	17	6	18	2	2	6	3	7	5	4	5	15	1	1	1	1	6	13	11

B) Mononucleotide model results

A	4	4	5	0	0	0	4	15	7	5	1	8	1	4	0	1	14	2	13	5	9	7
C	1	0	2	2	3	2	0	2	4	5	3	0	4	3	1	14	0	14	1	5	0	1
G	2	4	5	1	11	0	12	0	2	4	6	3	9	4	4	0	3	0	1	1	0	0
T	10	9	5	14	3	15	1	0	4	3	7	6	3	6	12	2	0	1	2	6	8	9

Fig. 1. CRP experimentally determined sites from the 18 loci listed in Figure 2. Although not all these sites were identified by footprinting experiments, to facilitate the presentation we refer to them in the text as footprint sites. A is compiled from these experimentally determined sites. B is compiled from the results of the mononucleotide model.

locations of the sites in each sequence, and a model, say, the monoresidue model, the probabilities at each position in the site can be estimated. The following generalization of Stormo's and Hartzell's information measure, the log likelihood, forms the basis for the required maximum likelihood estimates:

$$\log L = N\sum_{j=1}^{J}\sum_{b=A}^{T} f_{b,j}\log_e(\rho_{b,j}) + N(L-J)\sum_{b=A}^{T} f_{b,0}\log_e(\rho_{b,0}), \quad (1)$$

where $\rho_{b,O}$ are the unknown population base probabilities, parameters, for all positions outside of the site; $f_{b,O}$ are observed base frequencies; $n_{b,O}$ are the base counts outside the site; and $\rho_{b,j}$ are the parameters and $f_{b,j}$, are the base frequencies and $n_{b,j}$ the base counts for each position in the site. Note that Equation 1 differs from previous information measures used for this problem in two ways: 1) it is a generalization, since arbitary models for base frequencies may be employed, rather than only position-specific base frequency models; 2) it encompases all the data in each sequence, not just the data in the site. Thus the second term is added to the first to describe bases not in the site. As a result, if one of the features that charterize the site is a tendency for a different overall base composition than the rest of the sequence, this effect will be exploited by improvements in the second term of the log likelihood. In a sample of N segments of length L, each position within the site will yield N observed bases, but data from all (L − J) nonsite positions yeild N(L − J) nonsite observations. To obtain the maximum likelihood estimates, we find the values for the parameter estimates $\hat{\rho}_{b,j}$ that maximize the log likelihood. These are easily obtained in this case, since the values that maximize log L are the sample frequencies, i.e.

$$\hat{\rho}_{b,0} = f_{b,0} = n_{b,0}/(N[L-J])$$
$$\hat{\rho}_{b,j} = f_{b,j} = n_{b,j}/N. \quad (2)$$

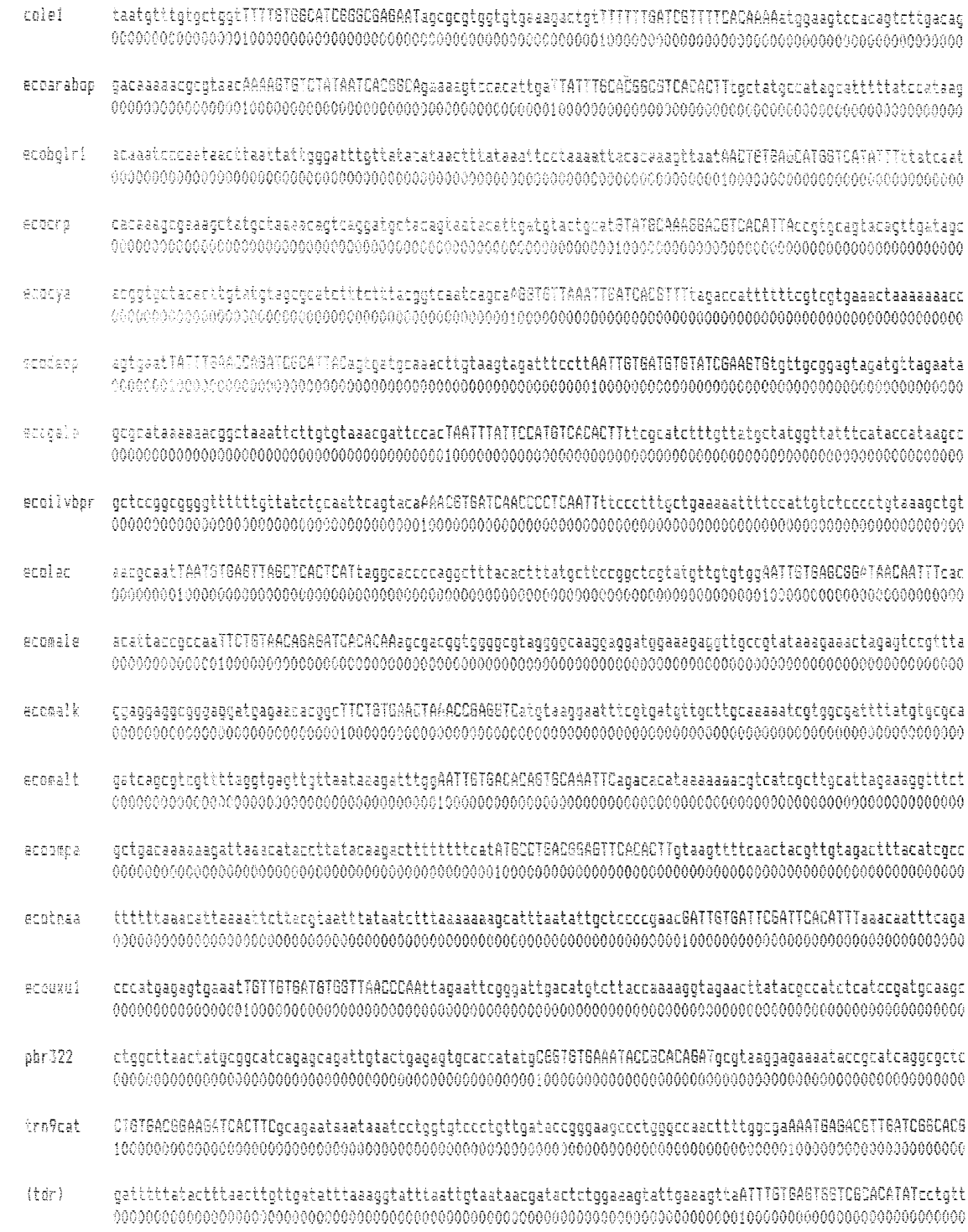


Fig. 2. Sequences from the 18 loci that contain CRP binding sites and site location vectors. The first 17 sequence are taken directly from Genbank rev 55. In new revisions of Genbank eco lmale and eco lmalk are combined in eco malba. (tdc) is not in Genbank; it was taken directly from Stormo and Hartzell.[9] Sequences have been chosen to place the sites at random positions.

These estimate are not available when information on the positions of the sites is missing; the $n_{b,j}$ are unknown.

EM algorithms are named for their two iterative steps, the expectation (E) step and the maximization (M) step, which are alternately repeated until a convergence criterion is satisfied. In the following description, we assume that we have completed some number of these iterative cycles, say (q − 1).

E Step

The site, which is J = 22 bases long, can start at any of L − J + 1 = 84 positions. We are missing the information that specifies the location of the start of each site. However, at the beginning of the E step of the q^{th} iteration, the current values of the population frequency estimates from the previous iteration of the M step are available. These values taken together specify the current estimate of the model parameters. With these values, we calculate the probability of observing the data in each sequence assuming that the site starts in each of the possible L − J + 1 (84) positions. These probabilities can now be used to calculate the probability that the site starts in each of the possible positions by using Bayes formula. Appendix A gives the formulas for this calculation. #5

Now, using the probabilities that the site starts in each of the possible positions as weights, add across the positions to find the expected number of the bases at each position in the site. For example, assume that there is an A in the first position of the window that starts at position 50 of, say, the third sequence. If the probability that the site starts at position 50 in the third sequence is 0.01: add 0.01 A's to the accumulating expected number of A's in the first position of the site. These expected values may be formally represented as follows:

$$\epsilon_{b,j}^{(q)} = E(n_{b,j} | \rho^{(q-1)}, S);$$
$$j = 0, \ldots, J; \; b = A, C, G, T, \qquad (3)$$

where S indicates the N available sequences.

M Step

Recall from Equation 2 that the maximum likelihood estimates for the population frequencies are just the sample frequencies when complete data are available. In the M step, substitute the expected number of bases for each position in the site from the E step for the unavailable directly observed number of bases into Equation 2:

$$\hat{\rho}_{b,j}^{(q)} = {}^{(q)}\epsilon_{b,j}/N \qquad j = 1, \ldots, J$$
$$b = A, C, G, T \qquad (4)$$
$$\hat{\rho}_{b,0}^{(q)} = \epsilon_{b,0}^{(q)}/(N[L - J]) \qquad b = A, C, G, T.$$

The algorithm converges when the parameter estimates stop changing, i.e., when

$$\hat{\rho}^{(q)} = \hat{\rho}^{(q-1)} = \hat{\rho}^{(*)}. \qquad (5)$$

At convergence the algorithm yields estimates of the population base probabilities, $\hat{\rho}_{b,j}$, and a set of posterior probabilities that indicate the probability that the site is at each of the possible positions in each sequence.

Motif Characterization

Although we have employed the mononucleotide model to illustrate the formulation of the algorithm, the algorithm encompasses a much broader set of motifs. To incorporate alternate motifs in our analysis, we consider alternate models for the population base frequencies for the positions within the site. For example, when a palindromic sequence is appropriate, expected numbers in the palindrome including bases, and their reverse complements are employed in Equations 3 and 4. Variable-length gaps within the site can be added to the algorithm through the use of a second missing variable, the gap length. This will result in the modification of the E step by including as candidate sites all locations with all allowed gaps lengths. #6 #7

When the binding motif is known a priori, then we need employ only the single model that corresponds to this motif. However, when the binding motif is not fully specified, the approach proposed here provides a means for choosing between the candidate motifs: We progressively impose more restrictions to yield a set of progressively more specific models. This sequential process is terminated when added restrictions reduce our ability to predict the data more than would be expected from chance alone. The likelihood ratio statistic is used to assess the limits of chance variation.

Application to CRP: The Data

We tested the algorithm by using CRP binding sites as an example. Beginning the analysis with only the assumption that CRP is a prokaryotic dimeric DNA binding protein yields a set of specific hypotheses about the binding motif. A palindromic binding motif at the positions in adjacent major groove openings implies a probability model specifying reverse complementarity of the bases separated by about six positions for the intervening minor groove. This model is implemented by requiring that the bases at positions 4–8 be the same as the probability of corresponding complementary bases in positions 19–15 (denoted in Table II as palindromic motifs). For example, the probability of a T at position 6 should match the probability of an A at position 17.

The bases in the minor groove that intervenes between the two major groove openings in the site are less accessible to the CRP protein. It has been suggested that the interaction is such that, at most, double-bonded base pairs (AT/TA) can be distin-

frequency estimation, segregation analysis and pedigree data analysis (Ceppellini, Siniscalco and Smith, 1955; Smith, 1957; Ott, 1979). A precursor to the broad recognition of the EM algorithm by the computational biology community is Churchill (1989), who applied the EM algorithm to fit a hidden Markov model (HMM) for partitioning genomic sequences into regions with homogenous base compositions. Lawrence and Reilly (1990) first introduced the EM algorithm for biological sequence motif discovery. Haussler et al. (1993) and Krogh et al. (1994) formulated an innovative HMM and used the EM algorithm for protein sequence alignment. Krogh, Mian and Haussler (1994) extended these algorithms to predict genes in *E. coli* DNA data. During the past two decades, probabilistic modeling and the EM algorithm have become a more and more common practice in computational biology, ranging from multiple sequence alignment for a single protein family (Do et al., 2005) to genome-wide predictions of protein–protein interactions (Deng et al., 2002), and to single-nucleotide polymorphism (SNP) haplotype estimation (Kang et al., 2004).

As noted in Meng and Pedlow (1992) and Meng (1997), there are too many EM-related papers to track. This is true even within the field of computational biology. In this paper we only examine a few key topics in computational biology and use typical examples to show how the EM algorithm has paved the road for these studies. The connection between the EM algorithm and statistical modeling of complex systems is essential in computational biology. It is our hope that this brief survey will stimulate further EM applications and provide insight for the development of new algorithms.

Discrete sequence data and continuous expression data are two of the most common data types in computational biology. We discuss sequence data analysis in Sections 2–5, and gene expression data analysis in Section 6. A main objective of computational biology research surrounding the "central dogma" is to study how the gene sequences affect the gene expression. In Section 2 we attempt to find conserved patterns in functionally related gene sequences as an effort to explain the relationship of their gene expression. In Section 3 we give an EM algorithm for multiple sequence alignment, where the goal is to establish "relatedness" of different sequences. Based on the alignment of evolutionary related DNA sequences, another EM algorithm for detecting potentially expression-related regions is introduced in Section 4. An alternative way to deduce the relationship between gene sequence and gene expression is to check the effect of sequence variation within the population of a species. In Section 5 we provide an EM algorithm to deal with this type of small sequence variation. In Section 6 we review the clustering analysis of microarray gene-expression data, which is important for connecting the phenotype variation among individuals with the expression level variation. Finally, in Section 7 we discuss trends in computational biology research.

2. SEQUENCE MOTIF DISCOVERY AND GENE REGULATION

In order for a gene to be transcribed, special proteins called transcription factors (TFs) are often required to bind to certain sequences, called transcription factor binding sites (TFBSs). These sites are usually 6–20 bp long and are mostly located upstream of the gene. One TF is usually involved in the regulation of many genes, and the TFBSs that the TF recognizes often exhibit strong sequence specificity and conservation (e.g., the first position of the TFBSs is likely T, etc.). This specific pattern is called a TF binding motif (TFBM). For example, Figure 1 shows a motif of length 6. The motif is represented by the position-specific frequency matrix $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_6)$, which is derived from the alignment of 5 motif sites by calculating position-dependent frequencies of the four nucleotides.

In order to understand how genes' mRNA expression levels are regulated in the cell, it is crucial to identify TFBSs and to characterize TFBMs. Although much progress has been made in developing experimental techniques for identifying these TFBSs, these techniques are typically expensive and time-consuming. They are also limited by experimental conditions, and cannot pinpoint the binding sites exactly. In the past twenty years, computational biologists and statisticians have developed many successful in silico methods to aid biologists in finding TFBSs, and these efforts have contributed significantly to our understanding of transcription regulation.

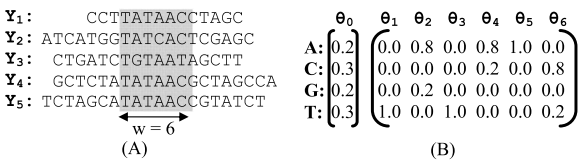


FIG. 1. *Transcription factor binding sites and motifs*. (A) *Each of the five sequences contains a TFBS of length* 6. *The local alignment of these sites is shown in the gray box*. (B) *The frequency of the nucleotides outside of the gray box is shown as* $\boldsymbol{\theta}_0$. *The frequency of the nucleotides in the ith column of the gray box is shown as* $\boldsymbol{\theta}_i$.

Likewise, motif discovery for protein sequences is important for identifying structurally or functionally important regions (domains) and understanding proteins' functional components, or active sites. For example, using a Gibbs sampling-based motif finding algorithm, Lawrence et al. (1993) was able to predict the key *helix-turn-helix* motif among a family of transcription activators. Experimental approaches for determining protein motifs are even more expensive and slower than those for DNAs, whereas computational approaches are more effective than those for TFBSs predictions.

The underlying logic of computational motif discovery is to find patterns that are "enriched" in a given set of sequence data. Common methods include word enumeration (Sinha and Tompa, 2002; Hampson, Kibler and Baldi, 2002; Pavesi et al., 2004), position-specific frequency matrix updating (Stormo and Hartzell, 1989; Lawrence and Reilly, 1990; Lawrence et al., 1993) or a combination of the two (Liu, Brutlag and Liu, 2002). The word enumeration approach uses a specific consensus word to represent a motif. In contrast, the position-specific frequency matrix approach formulates a motif as a weight matrix. Jensen et al. (2004) provide a review of these motif discovery methods. Tompa et al. (2005) compared the performance of various motif discovery tools. Traditionally, researchers have employed various heuristics, such as evaluating excessiveness of word counts or maximizing certain information criteria to guide motif finding. The EM algorithm was introduced by Lawrence and Reilly (1990) to deal with the motif finding problem.

As shown in Figure 1, suppose we are given a set of K sequences $\mathbf{Y} \equiv (\mathbf{Y}_1, \ldots, \mathbf{Y}_K)$, where $\mathbf{Y}_k \equiv (Y_{k,1}, \ldots, Y_{k,L_k})$ and $Y_{k,l}$ takes values in an alphabet of d residues ($d = 4$ for DNA/RNA and 20 for protein). The alphabet is denoted by $\mathbf{R} \equiv (r_1, \ldots, r_d)$. Motif sites in this paper refer to a set of contiguous segments of the same length w (e.g., the marked 6-mers in Figure 1). This concept can be further generalized via a hidden Markov model to allow gaps and position deletions (see Section 3 for HMM discussions). The weight matrix, or *Product-Multinomial* motif model, was first introduced by Stormo and Hartzell (1989) and later formulated rigorously in Liu, Neuwald and Lawrence (1995). It assumes that, if $Y_{k,l}$ is the ith position of a motif site, it follows the multinomial distribution with the probability vector $\boldsymbol{\theta}_i \equiv (\theta_{i1}, \ldots, \theta_{id})$; we denote this model as $PM(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_w)$. If $Y_{k,l}$ does not belong to any motif site, it is generated independently from the multinomial distribution with parameter $\boldsymbol{\theta}_0 \equiv (\theta_{01}, \ldots, \theta_{0d})$.

Let $\boldsymbol{\Theta} \equiv (\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_w)$. For sequence $\mathbf{Y}_k$, there are $L'_k = L_k - w + 1$ possible positions a motif site of length w may start. To represent the motif locations, we introduce the unobserved indicators $\boldsymbol{\Gamma} \equiv \{\Gamma_{k,l} \mid 1 \le k \le K, 1 \le l \le L'_k\}$, where $\Gamma_{k,l} = 1$ if a motif site starts at position l in sequence $\mathbf{Y}_k$, and $\Gamma_{k,l} = 0$ otherwise. As shown in Figure 1, it is straightforward to estimate $\boldsymbol{\Theta}$ if we know where the motif sites are. The motif location indicators $\boldsymbol{\Gamma}$ are the missing data that makes the EM framework a natural choice for this problem. For illustration, we further assume that there is exactly one motif site within each sequence and that its location in the sequence is uniformly distributed. This means that $\sum_l \Gamma_{k,l} = 1$ for all k and $P(\Gamma_{k,l} = 1) = \frac{1}{L'_k}$.

Given $\Gamma_{k,l} = 1$, the probability of each observed sequence $\mathbf{Y}_k$ is

$$P(\mathbf{Y}_k \mid \Gamma_{k,l} = 1, \boldsymbol{\Theta}) = \boldsymbol{\theta}_0^{\mathbf{h}(\mathbf{B}_{k,l})} \prod_{j=1}^{w} \boldsymbol{\theta}_i^{\mathbf{h}(\mathbf{Y}_{k,l+j-1})}. \tag{1}$$

In this expression, $\mathbf{B}_{k,l} \equiv \{Y_{k,j} : j < l \text{ or } j \ge l + w\}$ is the set of letters of nonsite positions of $\mathbf{Y}_k$. The counting function $\mathbf{h}(\cdot)$ takes a set of letter symbols as input and outputs the column vector $(n_1, \ldots, n_d)^T$, where n_i is the number of base type r_i in the input set. We define the vector power function as $\boldsymbol{\theta}_i^{\mathbf{h}(\cdot)} \equiv \prod_{j=1}^{d} \theta_{ij}^{n_j}$ for $i = 0, \ldots, w$. Thus, the complete-data likelihood function is the product of equation (1) for k from 1 to K, that is,

$$\begin{aligned} P(\mathbf{Y}, \boldsymbol{\Gamma} \mid \boldsymbol{\Theta}) &\propto \prod_{k=1}^{K} \prod_{l=1}^{L'_k} P(\mathbf{Y}_k \mid \Gamma_{k,l} = 1, \boldsymbol{\Theta})^{\Gamma_{k,l}} \\ &= \boldsymbol{\theta}_0^{\mathbf{h}(\mathbf{B}_{\boldsymbol{\Gamma}})} \prod_{i=1}^{w} \boldsymbol{\theta}_i^{\mathbf{h}(\mathbf{M}_{\boldsymbol{\Gamma}}^{(i)})}, \end{aligned}$$

where $\mathbf{B}_{\boldsymbol{\Gamma}}$ is the set of all nonsite bases, and $\mathbf{M}_{\boldsymbol{\Gamma}}^{(i)}$ is the set of nucleotide bases at position i of the TFBSs given the indicators $\boldsymbol{\Gamma}$.

The MLE of $\boldsymbol{\Theta}$ from the complete-data likelihood can be determined by simple counting, that is,

$$\hat{\boldsymbol{\theta}}_i = \frac{\mathbf{h}(\mathbf{M}_{\boldsymbol{\Gamma}}^{(i)})}{K} \quad \text{and} \quad \hat{\boldsymbol{\theta}}_0 = \frac{\mathbf{h}(\mathbf{B}_{\boldsymbol{\Gamma}})}{\sum_{k=1}^{K}(L_k - w)}.$$

The EM algorithm for this problem is quite intuitive. In the E-step, one uses the current parameter values $\boldsymbol{\Theta}^{(t)}$ to compute the expected values of $h(\mathbf{M}_{\boldsymbol{\Gamma}}^{(i)})$ and $h(\mathbf{B}_{\boldsymbol{\Gamma}})$. More precisely, for sequence Y_k, we compute its likelihood of being generated from $\boldsymbol{\Theta}^{(t)}$ conditional on each

possible motif location $\Gamma_{k,l} = 1$,

$$w_{k,l} \equiv P\big(\mathbf{Y}_k|\Gamma_{k,l} = 1, \boldsymbol{\Theta}^{(t)}\big) = \left(\frac{\boldsymbol{\theta}_1}{\boldsymbol{\theta}_0}\right)^{\mathbf{h}(Y_{k,l})} \cdots \left(\frac{\boldsymbol{\theta}_w}{\boldsymbol{\theta}_0}\right)^{\mathbf{h}(Y_{k,l+w-1})} \boldsymbol{\theta}_0^{\mathbf{h}(\mathbf{Y}_k)}.$$

Letting $W_k \equiv \sum_{l=1}^{L'_k} w_{k,l}$, we then compute the expected count vectors as

$$E_{\boldsymbol{\Gamma}|\boldsymbol{\Theta}^{(t)},\mathbf{Y}}\big[\mathbf{h}\big(\mathbf{M}_{\boldsymbol{\Gamma}}^{(i)}\big)\big] = \sum_{k=1}^{K}\sum_{l=1}^{L'_k} \frac{w_{k,l}}{W_k}\mathbf{h}(Y_{k,l+i-1}),$$

$$E_{\boldsymbol{\Gamma}|\boldsymbol{\Theta}^{(t)},\mathbf{Y}}[\mathbf{h}(\mathbf{B}_{\boldsymbol{\Gamma}})] = \mathbf{h}(\{Y_{k,l} : 1 \le k \le K, 1 \le l \le L_k\}) - \sum_{i=1}^{w} E_{\boldsymbol{\Gamma}|\boldsymbol{\Theta}^{(t)},\mathbf{Y}}\big[\mathbf{h}\big(\mathbf{M}_{\boldsymbol{\Gamma}}^{(i)}\big)\big].$$

In the M-step, one simply computes

$$\boldsymbol{\theta}_i^{(t+1)} = \frac{E_{\boldsymbol{\Gamma}|\boldsymbol{\Theta}^{(t)},\mathbf{Y}}\big[\mathbf{h}\big(\mathbf{M}_{\boldsymbol{\Gamma}}^{(i)}\big)\big]}{K} \quad \text{and}$$

$$\boldsymbol{\theta}_0^{(t+1)} = \frac{E_{\boldsymbol{\Gamma}|\boldsymbol{\Theta}^{(t)},\mathbf{Y}}[\mathbf{h}(\mathbf{B}_{\boldsymbol{\Gamma}})]}{\sum_{k=1}^{K}(L_k - w)}.$$

It is necessary to start with a nonzero initial weight matrix $\boldsymbol{\Theta}^{(0)}$ so as to guarantee that $P(\mathbf{Y}_k|\Gamma_{k,l} = 1, \boldsymbol{\Theta}^{(t)}) > 0$ for all l. At convergence the algorithm yields both the MLE $\hat{\boldsymbol{\Theta}}$ and predictive probabilities for candidate TFBS locations, that is, $P(\Gamma_{k,l} = 1|\hat{\boldsymbol{\Theta}}, \mathbf{Y})$.

Cardon and Stormo (1992) generalized the above simple model to accommodate insertions of variable lengths in the middle of a binding site. To overcome the restriction that each sequence contains exactly one motif site, Bailey and Elkan (1994, 1995a, 1995b) introduced a parameter p_0 describing the prior probability for each sequence position to be the start of a motif site, and designed a modified EM algorithm called the Multiple EM for Motif Elicitation (MEME). Independently, Liu, Neuwald and Lawrence (1995) presented a full Bayesian framework and Gibbs sampling algorithm for this problem. Compared with the EM approach, the Markov chain Monte Carlo (MCMC)-based approach has the advantages of making more flexible moves during the iteration and incorporating additional information such as motif location and orientation preference in the model.

The generalizations in Bailey and Elkan (1994) and Liu, Neuwald and Lawrence (1995) assume that all overlapping subsequences of length w in the sequence data set are from a finite mixture model. More precisely, each subsequence of length w is treated as an independent sample from a mixture of $PM(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_w)$ and $PM(\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_0)$ [independent Multinomial(θ_0) in all w positions]. The EM solution of this mixture model formulation then leads to the MEME algorithm of Bailey and Elkan (1994). To deal with the situation that w may not be known precisely, MEME searches motifs of a range of different widths separately, and then performs model selection by optimizing a heuristic function based on the maximum likelihood ratio test. Since its release, MEME has been one of the most popular motif discovery tools cited in the literature. The Google scholar search gives a count of 1397 citations as of August 30th, 2009. Although it is 15 years old, its performance is still comparable to many new algorithms (Tompa et al., 2005).

3. MULTIPLE SEQUENCE ALIGNMENT

Multiple sequence alignment (MSA) is an important tool for studying structures, functions and the evolution of proteins. Because different parts of a protein may have different functions, they are subject to different selection pressures during evolution. Regions of greater functional or structural importance are generally more conserved than other regions. Thus, a good alignment of protein sequences can yield important evidence about their functional and structural properties.

Many heuristic methods have been proposed to solve the MSA problem. A popular approach is the progressive alignment method (Feng and Doolittle, 1987), in which the MSA is built up by aligning the most closely related sequences first and then adding more distant sequences successively. Many alignment programs are based on this strategy, such as MULTALIGN (Barton and Sternberg, 1987), MULTAL (Taylor, 1988) and, the most influential one, ClustalW (Thompson, Higgins and Gibson, 1994). Usually, a *guide tree* based on pairwise similarities between the protein sequences is constructed prior to the multiple alignment to determine the order for sequences to enter the alignment. Recently, a few new progressive alignment algorithms with significantly improved alignment accuracies and speed have been proposed, including T-Coffee (Notredame, Higgins and Heringa, 2000), MAFFT (Katoh et al., 2005), PROBCONS (Do et al., 2005) and MUSCLE (Edgar, 2004a, 2004b). They differ from previous approaches and each other mainly in the construction of the guide tree and in the objective function for judging the goodness of the alignment. Batzoglou (2005) and Wallace, Blackshields and Higgins (2005) reviewed these algorithms.

SujetPratique_PartielAlgoSto_Novembre2021

November 4, 2021

1 Sujet pratique

L'objectif de ce sujet est d'implémenter diverses variantes de l'algorithme de descente de gradient stochastique (SGD) vu en cours. On s'appuiera sur l'article Algorithmes de Descente de Gradient Stochastique avec le filtrage des paramètres pour l'entraînement des réseaux à convolution profonds dont vous avez une copie papier.

On supposera que les données sont des couples (x, y) avec x et y deux nombres réels, et que les paramètres sont de la forme (θ_1, θ_2) où θ_1, θ_2 sont deux nombres réels.

Les données seront représentées par deux tableaux (`numpy.array`) `X` et `Y` de même longueur.

On supposera fixée une fonction de coût $J : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ telle que $J((x, y), (\theta_1, \theta_2))$ renvoie le "coût" associé aux données (x, y) pour les paramètres (θ_1, θ_2) d'une certaine boîte noire (ici un réseau profond qu'on ne cherchera pas à construire). On supposera que J a toute la régularité nécessaire.

En TP, vous avez implémenté une descente de gradient stochastique pour minimiser la fonction J. La partie 2.2. de l'article en présente plusieurs variantes. **Attention, il s'agit bien d'algorithmes stochastiques, même si la présentation ne le met en évidence qu'une seule fois dans l'équation (2).**

Vous disposez d'une fonction **gradJ** qui calcule le gradient de J par rapport aux paramètres θ_1, θ_2. Elle se présente sous la forme suivante :

```
def gradJ(x,y,theta1,theta2):
    # snip #
    return (partial1, partial2)
```

Vous n'avez **pas** à coder cette fonction vous-mêmes. Vous pouvez appeler **gradJ** dans votre code, en respectant la syntaxe. Pour l'instant, afin que le code puisse être validé sans erreur, on va définir une fonction triviale qui n'a aucun sens concret.

[2]:
```
def gradJ(x,y,theta1,theta2):
    return (0,0)
```

1. Coder une fonction `momentum_sgd` qui prend pour arguments :
   - Un nombre `n_iter` d'itérations
   - Un pas de temps `eta` et un paramètre `mu`
   - Des entrées `X`, `Y` comme décrites ci-dessus

et renvoie la valeur du paramètre θ après `n_iter` itérations de l'algorithme de "Momentum SGD" tel que décrit dans la section 2.2. Attention encore une fois, l'algorithme est **stochastique**. Votre fonction doit renvoyer un couple de réels.

[7]:
```
def momentum_sgd(n_iter, eta, mu, X, Y):
    ## TODO ##
    return (theta1, theta2)
```

2. Coder une fonction `momentom_sg` avec les mêmes arguments. Cette fois vous implémenterez l'algorithme de "descente de gradient accélérée de Nesterov" **dans sa version stochastique**.

[6]:
```
def momentom_sgd(n_iter, eta, mu, X, Y):
    ## TODO ##
    return (theta1, theta2)
```

3. Coder une fonction `adagrad` qui prend pour arguments :
   - Un nombre `n_iter` d'itérations
   - Un paramètre `alpha`
   - Des entrées `X, Y` comme décrites ci-dessus

   et renvoie la valeur du paramètre θ après `n_iter`itérations de l'algorithme "Adagrad" tel que décrit dans la section 2.3. Attention, cette fois-ci, implémentez une descente de gradient **déterministe**.

[9]:
```
def adagrad(n_iter, alpha, X, Y):
    # TODO #
    return (theta1, theta2)
```

4. Coder une fonction `adam` qui prend pour arguments :
   - Un nombre `n_iter` d'opérations
   - Des paramètres `beta1`, `beta2`, `alpha`, `epsilon`
   - Des entrées `X, Y` comme décrites ci-dessus

   et implémente l'algorithme "Adam" tel que décrit dans la section 2.3. Attention, on est de retour dans un cadre **stochastique**.

[8]:
```
def adam(n_iter, beta1, beta2, alpha, epsilon, X, Y):
    ## TODO ##
    return (theta1, theta2)
```

5. Enfin, implémentez la "méthode 1" (moyenne pondérée) décrite dans la section 3.1 de l'article. Attention, les λ_u de l'équation (10) doivent être correctement normalisés, mais vous pouvez ignorer ce détail si besoin. Il s'agit encore une fois d'une descente **stochastique**.

[10]:
```
def moy_pond(n_iter, eta, mu, delta, sigma, X, Y):
    ## TODO ##
    return (theta1, theta2)
```

7. Si vous avez fini, vous pouvez vous amusez à 1) générer des données 2) écrire une "vraie" fonction de coût pour un problème de classification (comme par exemple celle du TP) et faire tourner les algorithmes précédents sur un exemple.

Algorithmes de Descente de Gradient Stochastique avec le filtrage des paramètres pour l'entraînement des réseaux à convolution profonds

Pierre Gillot[1] Akka Zemmari[1] Jenny Benois-Pineau[1] Yurii Nesterov[2]

[1] Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France
[2] CORE, Catholic University of Louvain (UCL), Belgium

Résumé

Dans cet article, nous nous intéressons à la définition de nouveaux algorithmes d'apprentissage profond afin d'améliorer la qualité et la stabilité de l'entrainement des paramètres des réseaux de neurones à convolution. Notre contribution se présente sous forme de deux algorithmes, basés tous les deux sur le filtrage des paramètres du réseau de neurones. Les expériences menées montrent qu'un réglage fin de leurs (hyper) paramètres améliore clairement la qualité et la stabilité des résultats tant en phase d'entrainement qu'en phase de validation.

Mots Clef

Apprentissage profond, optimisation, descente de gradient.

1 Introduction

Les **réseaux de neurones artificiels** (**ANN**) sont des algorithmes inspirés de la biologie, capables *d'apprendre* à réaliser une tâche de classification. Un réseau de neurones est composé d'un ensemble de couches successives. Chaque couche est composée des neurones artificiels [1]. Un neurone de la $k^{\text{ième}}$ couche est connecté à tous les neurones de la $(k-1)^{\text{ième}}$. Entrainer un réseau revient alors à apprendre les poids synaptiques et les biais de chaque neurone.

Les réseaux de neurones présentent un problème de passage à l'échelle lorsqu'ils sont utilisés pour la classification d'ensembles importants de données en particulier d'images. Leur architecture de base, où l'entrée de chaque neurone d'une couche est connectée à toutes les sorties des neurones de la couche précédente, rend l'apprentissage du nombre trop grand de paramètres non envisageable.

Les **réseaux de neurones à convolution** (**CNN**) ont été conçus pour la classification d'images. Leur architecture est différente de celle des réseaux ANN. Les couches ne sont pas toutes de même nature : couches de convolution, transformation non linéaire, couches de *pooling*, etc. Les neurones ne sont plus connectés à tous les neurones de couches précédentes, ... Dans la plupart d'architectures des CNN profonds, l'apprentissage des paramètres se fait par l'algorithme de *descente de gradient stochastique* (SGD) [1].

Les méthodes de descente de gradient sont efficaces pour les problèmes convexes. Cependant, la fonction de perte (que l'on cherche à minimiser dans la phase d'apprentissage) n'est pas toujours convexe. Par expérience, on remarque qu'il est nécessaire d'effectuer un grand nombre d'itérations avec une grande instabilité de cette fonction. Dans cet article, nous explorons les méthodes d'optimisation basées sur la SGD. Nous proposons des méthodes d'optimisation (*moyenne pondérée* et *moyenne glissante pondérée*) permettant de stabiliser le processus d'entrainement. Nous étudions ces méthodes de manière expérimentale et ce sur un corpus de données image de référence, le corpus *MNIST* [2].

2 Travaux antérieurs

2.1 Apprentissage et optimisation dans les réseaux profonds

Descente de gradient. La méthode de descente de gradient a fait ses preuves pour l'optimisation des paramètres d'un réseau de neurones. Elle est ainsi l'une des approches les plus efficaces et les plus utilisées. Nous rappelons brièvement quelques notions sur cette méthode. Plus de détails peuvent être trouvés dans [1].

Soit θ le vecteur contenant tous les paramètres de notre réseau CNN. Soit $J\big(\theta, g(x), y\big)$ la fonction de coût représentant l'erreur entre la vérité terrain $g(x)$ (liée à la donnée d'entrainement x) et la valeur prédite y estimée par le réseau en utilisant les paramètres θ.

Un pas de l'itération de la descente de gradient est donné par :

$$\theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta J\big(\theta, g(x), y\big), \tag{1}$$

où η est une constante positive appelée *taux d'apprentissage*. La méthode s'applique pour tout espace de dimension quelconque (même infinie) et peut être utilisée aussi bien pour les fonctions linéaires que pour les non linéaires. L'algorithme ne converge vers la solution globale que si la fonction J est strictement convexe dans le domaine d'optimisation des paramètres.

En apprentissage profond, le nombre de paramètres à apprendre devient très vite important. Les arguments de la fonction de coût sont alors de très grande dimension, on

observe alors une prolifération de points-selles, lesquels peuvent être très pénalisant quand on cherche un minimum car ils sont, en général, entourés de plateaux [3].

D'autre part, le choix du taux d'apprentissage est crucial : il doit être suffisamment petit pour "assurer" la convergence de l'algorithme mais également pas trop petit pour ne pas trop retarder le temps de convergence du processus d'optimisation. Le choix d'un taux par itération peut alors améliorer considérablement les résultats. L'algorithme dit *line search* [4] est une solution classique à ce problème.

2.2 Descente de gradient stochastique.

En apprentissage profond, la fonction objective que l'on cherche à minimiser est souvent non convexe et non régulière. La convergence de la descente du gradient vers le minimum global n'est donc pas garantie et la convergence même vers un minimum local peut être extrêmement lente. Une solution à ce problème consiste en l'utilisation de l'algorithme de descente de gradient stochastique. L'idée de l'approche est de chercher à minimiser une fonction qui peut être écrite sous la forme de la somme de fonctions différentiables. Ce processus est alors réalisé de manière itérative sur des lots de données tirés aléatoirement. Chaque fonction objective minimisée de cette manière est une approximation de la fonction objective globale. L'équation suivante décrit cette méthode :

$$\theta_{t+1} \leftarrow \theta_t - \eta \frac{1}{B_s} \sum_{i=1}^{B_s} \nabla_\theta J_t^i(\theta_t, x_t^i, y_t^i), \tag{2}$$

où B_s est la taille du lot, $B_t = (x_t^i, y_t^i)_{i \in [|1, B_s|]}$ est le lot de données tiré à l'étape t, x_t^i et y_t^i sont respectivement la donnée de la vérité terrain et la donnée estimée dans ce lot, et $J_t = \frac{1}{B_s} \sum_{i=1}^{B_s} J_t^i$ est l'approximation stochastique de la fonction de coût globale à l'étape t sur le lot B_t, décomposée en somme de fonctions différentiables J_t^i liées à chaque paire (x_t^i, y_t^i). La Descente de gradient stochastique (SGD) résout la plupart des problèmes rencontrés en apprentissage profond.

Momentum SGD. L'objectif principal de la méthode dite de *Momentum* est d'accélérer le processus de descente de gradient, et ceci en rajoutant un vecteur de vélocité à l'expression initiale :

$$\begin{aligned} v_{t+1} &\leftarrow \mu v_t - \eta \nabla J(\theta_t), \\ \theta_{t+1} &\leftarrow v_{t+1} + \theta_t. \end{aligned} \tag{3}$$

Le vecteur v_{t+1} est calculé au début de chaque itération et représente la mise à jour de la vélocité d'une "balle dévalant une pente".

La vélocité s'accumule à chaque itération d'où l'introduction d'un hyper-paramètre μ permettant d'amortir la vélocité quand on atteint une surface plate. Une bonne stratégie peut être de modifier μ en fonction du niveau d'apprentissage.

Descente de gradient accélérée de Nesterov. En 1983, Nesterov proposa dans [5] une modification de la méthode du Momentum et montra que son algorithme présente une meilleure convergence théorique pour l'optimisation des fonctions convexes. Cette approche devint très populaire grâce à ses performances en pratique comparée à la méthode classique.

La principale différence entre la méthode de Nesterov et la méthode du momentum réside dans le fait que cette dernière commence par calculer le gradient à l'emplacement courant θ_t avant de faire un pas dans la direction de la vélocité accumulée, alors que le momentum de Nesterov fait d'abords un pas de calcul pour obtenir une approximation du paramètre mis à jour, que l'on note $\widetilde{\theta}_{t+1}$, et corrige ensuite ce pas en calculant le gradient à cet emplacement.

Un pas du momentom de Nestrov est décrit par :

$$\begin{aligned} \widetilde{\theta}_{t+1} &\leftarrow \theta_t + \mu v_t, \\ v_{t+1} &\leftarrow \mu v_t - \eta \nabla J\big(\widetilde{\theta}_{t+1}\big), \\ \theta_{t+1} &\leftarrow v_{t+1} + \theta_t. \end{aligned} \tag{4}$$

2.3 Extensions de la descente de gradient

Il existe plusieurs variantes de l'algorithme de descente de gradient. Dans la suite, nous présentons trois méthodes différentes assez proches de nos méthodes présentées dans ce travail :

Descente de gradient moyennée. Cette méthode a été proposée et étudiée par Polyak dans [6]. L'idée est de remplacer le calcul du paramètre θ_t, par le calcul de la moyenne temporelle de ces valeurs, et ce à partir des mises à jour obtenues par la descente du gradient :

$$\overline{\theta}_T \leftarrow \frac{1}{T} \sum_{t=0}^{T} \theta_t. \tag{5}$$

Adagrad. Le principe de cette méthode, proposée en 2011 dans [7], est de faire que le taux d'apprentissage s'adapte aux paramètres, faisant de sorte qu'il s'ajuste automatiquement, en fonction de "l'éparsité" des paramètres. Adagrad abaisse progressivement le taux d'apprentissage mais pas de la même manière pour tous les paramètres : les dimensions à pente plus prononcée voient leur taux abaissé plus rapidement que celles à pente douce. Plus formellement, le pas est décrit par :

$$\forall i,\ (\theta_{t+1})_i \leftarrow (\theta_t)_i - \alpha \frac{\big(\nabla J(\theta_t)\big)_i}{\sqrt{\sum_{u=1}^{t} \big(\nabla J(\theta_u)\big)_i^2}},\ \alpha > 0. \tag{6}$$

RMSProp L'algorithme [8] ajuste automatiquement le taux d'apprentissage à chaque paramètre, comme Adagrad. Cependant, il ne cumule que les gradients issus des itérations récentes. Pour cela, il utilise une moyenne glissante :

$$\begin{aligned} &\forall i,\ (\nabla_{t+1})_i \leftarrow \delta(\nabla_t)_i + (1-\delta)\big(\nabla J(\theta_t)\big)_i^2, \\ &\forall i,\ (\theta_{t+1})_i \leftarrow (\theta_t)_i - \alpha \frac{\big(\nabla J(\theta_t)\big)_i}{\sqrt{(\nabla_{t+1})_i}},\ \alpha > 0. \end{aligned} \tag{7}$$

Ici, $\delta(\nabla_t)_i$ est la moyenne quadratique glissante du gradient. La division du gradient de la fonction objective par la racine de la moyenne quadratique glissante (c'est à dire l'amplitude) améliore la convergence.

Adam. Adam [9] est l'un des algorithmes les plus récents et les plus efficaces pour l'optimisation par descente de gradient. Le principe est le même que pour Adagrad et RMSProp : il adapte automatiquement le taux d'apprentissage pour chaque paramètre. Sa particularité est de calculer (m_t, v_t) des "estimations adaptatives des moments". Il peut donc être vu comme une généralisation de l'algorithme Adagrad :

$$\begin{array}{l} \forall i,\ (m_{t+1})_i \leftarrow \beta_1 \cdot (m_t)_i + (1-\beta_1) \cdot \big(\nabla J(\theta_t)\big)_i, \\ \forall i,\ (v_{t+1})_i \leftarrow \beta_2 \cdot (v_t)_i + (1-\beta_2) \cdot \big(\nabla J(\theta_t)\big)_i^2, \\ \forall i,\ (\theta_{t+1})_i \leftarrow (\theta_t)_i - \alpha \frac{\sqrt{1-\beta_2}}{1-\beta_1} \frac{(m_t)_i}{\sqrt{(v_t)_i + \epsilon}}, \\ \alpha, \epsilon > 0 \text{ et } \beta_1, \beta_2 \in]0,1[. \end{array} \tag{8}$$

Ici m_t est le premier moment du gradient (la moyenne) et v_t est son second moment (variance non-centrée). ϵ est un paramètre de précision. Sa valeur par défaut dans l'outil populaire d'apprentissage des CNN Caffe [10] est 10^{-8}. Les paramètres β_1 et β_2 sont utilisés pour réaliser des moyennes d'exécution sur les moments m_t et v_t respectivement.

3 Notre contribution : SGD avec filtrage des paramètres

Dans ce travail, nous nous intéressons à la mise à jour du vecteur de paramètres et proposons des méthodes compatibles avec les solveurs existant (sous Caffe notamment). L'idée principale est qu'au lieu de ne considérer que la dernière mise à jour d'un paramètre, nous considérons des moyennes pondérées de ses dernières mises à jour. Cela permet de filtrer les paramètres du réseau assurant une meilleure convergence aussi bien en terme de "profondeur" de la descente que de la stabilité. L'objectif étant de diminuer la perte et d'augmenter la précision en moyenne tout en réduisant leurs variances. La stabilité du processus permet alors de réduire le nombre d'itérations.

A noter que vue le corpus considéré dans ce travail (MNIST), nous ne nous sommes pas intéressés aux temps d'exécution des différents algorithmes mais plutôt à leurs précisions.

3.1 Moyenne pondérée pour la descente de gradient

Notre première méthode de filtrage est appelée "Moyenne pondérée". L'idée générale de la méthode est de mettre à jour la valeur courante du paramètre en fonction de ses valeurs aux itérations précédentes et ce en utilisant la *mémoire* de l'entrainement. Le filtrage des paramètres du réseau peut s'opérer i) au début du processus d'optimisation (Méthode 1) ou encore ii) quand une stabilisation de la moyenne temporelle de la fonction objective est observée (Méthode 2). Dans la suite, nous avons choisi d'utiliser la méthode de momentum de Nesterov comme méthode de base. Cependant, il est clair que notre approche est compatible avec toutes les autres méthodes d'optimisation.

Méthode 1 : Moyenne pondérée. Nous introduisons un effet *mémoire* sur la mise à jour du paramètre de vélocité :

$$\begin{array}{l} \widetilde{\theta}_{t+1} \leftarrow \theta_t + \mu v_t,\ \ v_{t+1} \leftarrow \mu v_t - \eta \nabla J\big(\widetilde{\theta}_{t+1}\big), \\ \theta_{t+1} \leftarrow \left\{ \begin{array}{l} v_{t+1} + \theta_t,\ \forall t < \delta \\ v_{t+1} + \sum\limits_{u=t-\delta+1}^{t} \lambda_u \theta_u,\ \forall t \geq \delta \end{array} \right. \end{array} \tag{9}$$

où δ est le nombre de mises à jour passées utilisées pour effectuer une nouvelle mise à jour, et (λ_u) est une suite décroissante de nombres positifs, telle que $\sum\limits_{u=t-\delta+1}^{t} \lambda_u = 1$. Nous avons utilisé le cas gaussien pour cette suite, i.e., avant la normalisation :

$$\forall u \in [|t-\delta+1, t|],\ \lambda_u = e^{-\frac{(t-u)^2}{\sigma^2}},\ \sigma \in \mathbb{R}. \tag{10}$$

Ce nouveau modèle utilise deux hyper-paramètres δ et σ. Le paramètre δ permet de contrôler le nombre de mises à jour à prendre en compte dans un pas de mise à jour, le paramètre σ permet lui de régler la pondération des mises à jour passées. À noter que les poids sont de plus en plus forts pour les dernières mises à jour. Les valeurs choisies pour σ sont en cohérence avec celles prises par δ. Plus précisément, nous nous assurons que σ vérifie bien $\lambda_t = \kappa \lambda_{t-\delta+1}$. Les valeurs prises par δ sont comprises entre 4 et 32 et pour ces valeurs, nous choisissons $\kappa = 10$. Cela permet de s'assurer que les mises à jour les plus récentes seront privilégiées dans le processus de mise à jour.

Méthode 2 : Moyenne pondérée avec délai. Le principe de cette est méthode est de commencer la régularisation des paramètres en partant de fonctions cibles déjà stables. Le moment de stabilisation est défini par des variations limitées de la moyenne temporelle $m(t)$ de la fonction cible (fonction de perte de l'entrainement ou fonction de précision) : on choisit le premier t tel que le critère :

$$\frac{|m(t) - m(t-1)|}{m(t-1)} < \alpha \tag{11}$$

soit satisfait pour un α donné. Le temps auquel le moyennage commence est noté T_{ba}. L'expression de notre solveur est donnée par :

$$\begin{array}{l} \widetilde{\theta}_{t+1} \leftarrow \theta_t + \mu v_t,\ \ v_{t+1} \leftarrow \mu v_t - \eta \nabla J\big(\widetilde{\theta}_{t+1}\big), \\ \theta_{t+1} \leftarrow \left\{ \begin{array}{l} v_{t+1} + \theta_t,\ \forall t < \max\big\{\delta, T_{ba}\big\} \\ v_{t+1} + \sum\limits_{u=t-\delta+1}^{t} \lambda_u \theta_u,\ \forall t \geq \max\big\{\delta, T_{ba}\big\} \end{array} \right. \end{array} \tag{12}$$

3.2 Moyenne glissante pondérée avec délai

La Méthode 2 a montré, lors de nos expérimentations, une meilleure stabilité de la fonction objective (perte). Cependant, elle nécessite l'utilisation d'une mémoire-tampon