TP5: Réseaux récurrents

Gaétan Marceau Caron

21 janvier 2016

1 Introduction

Lors des séances précédentes, nous nous sommes intéressés à la tâche de la classification d'images avec un réseau neuronal classique. En particulier, nous n'avons pas pris en compte la topologie naturelle de l'image. Un réseau de convolution utilise directement cette information en parcourant l'image avec des noyaux qui traitent chaque pixel et son voisinage. En conséquence, cette information permet au réseau de convolution de dépasser significativement les réseaux classiques dans le traitement d'images.

De manière analogique, différent modèles issus des réseaux de neurones sont adaptés aux structures naturelles des données. Un exemple important consiste à modéliser la probabilité d'observer une séquence de symboles. De manière générale, les symboles ne sont pas tous mutuellement indépendants. Par exemple, dans les langages naturels, la probabilité d'une lettre dépend des lettres voisines.

Les réseaux récurrents sont des réseaux de neurones avec une mémoire permettant d'apprendre un contexte et de faire une prédiction. Ces réseaux sont adaptés pour l'apprentissage de séquences, c'est-à-dire de données possédant un ordre naturel. Par exemple, le texte, la vidéo ou l'enregistrement audio sont des séquences pouvant être traitées par des réseaux récurrents. [1] Le but du réseau récurrent est alors de prédire la prochaine donnée en apprenant un modèle stochastique. De plus, une fois entraîné, le réseau peut être utilisé comme modèle génératif.

Dans ce TP5, nous allons étudier un réseau récurrent simple pour la tâche de prédiction de caractères. Le modèle du langage que nous allons utiliser se définit comme :

$$p(x^1, \ldots, x^n) = \prod_{i=1}^{n} p(x^i | x^1, \ldots, x^{i-1}) \quad \text{(règle de chaînage)} \tag{1}$$

et nous allons modéliser cette loi de probabilité à l'aide d'un réseau récurrent.

Le but du réseau récurrent est de prédire le prochain caractère o^t en fonction des caractères observés $(x^1, \ldots, x^t)$. Pour modéliser les dépendances à long terme, le réseau est doté d'une mémoire interne h qui est modifiée au cours du

1. voir `http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/`

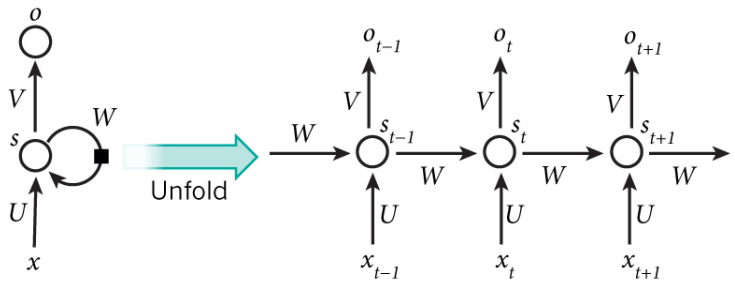

FIGURE 1 – Déroulement du réseau récurrent ($s = h, U = W_{xh}, V = W_{hy}, W = W_{hh}$) (source : http ://www.wildml.com)

temps (nous noterons ces valeurs h^t). Nous pouvons donc définir les équations de la procédure `forward` permettant de transformer les données en entrée :

$$h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{2}$$
$$y_t = W_{hy}h_t + b_y \tag{3}$$
$$p_t = \text{softmax}(y_t) \tag{4}$$
$$o_t \sim p_t \tag{5}$$

où σ est une fonction d'activation, W_{xh} est une matrice transformant les entrées en élément de la mémoire, W_{hh} est une matrice carrée propageant la mémoire du temps $t-1$ au temps t et W_{hy} est une matrice qui convertit la mémoire en sortie. De plus, nous utilisons une variable y^t pour les sorties non normalisées et p^t pour les sorties normalisées (la loi de probabilité apprise). Finalement, nous rappelons, sous la forme d'un pseudocode (cf. algorithme 1), l'algorithme «Backpropation Through Time» (BPTT).

2 Description

Pour ce TP, vous devez compléter le code fourni :

1. dans la procédure forward (`model.py::98-99`), implémenter la définition des équations 2 et 3
2. dans la procédure backward (`model.py:110-119`), implémenter l'algorithme BPTT (voir pseudocode 1)

Vous pouvez vérifier votre implémentation avec le test des différences finies `gradCheck:model.py:183`. Une fois l'implémentation vérifiée, vous pouvez commenter l'appel à `gradCheck` et admirer les premiers balbutiements de votre réseau récurrent !

Data: $(x^t, h^t, p^t)_{t=1}^n$
Result: $\delta W_{xh}, \delta W_{hh}, \delta W_{hy}, \delta b_h, \delta b_y$
$\delta W_{xh} = 0, \delta W_{hh} = 0, \delta W_{hy} = 0, \delta b_h = 0, \delta b_y = 0, \delta h^{next} = 0;$
for *t=n to 0* **do**
  $\delta y = p^t - e_{x^{t+1}}$;
  $\delta W_{hy} = \delta W_{hy} + \delta y \cdot (h^t)^\top;$
  $\delta b_y = \delta b_y + \delta y;$
  $\delta h = W_{hy}^\top \cdot \delta y + \delta h^{next};$
  $\delta h_{raw} = (1 - h^t \odot h^t) \odot \delta h;$
  $\delta b_h = \delta b_h + \delta h_{raw};$
  $\delta W_{xh} = \delta W_{xh} + \delta h_{raw} \cdot (x^t)^\top;$
  $\delta W_{hh} = \delta W_{hh} + \delta h_{raw} \cdot (h^{t-1})^\top;$
  $\delta h^{next} = W_{hh}^\top \cdot \delta h_{raw};$
end

Algorithm 1: Pseudocode de BPTT

3 Livrable

Date du livrable : avant le 29 janvier 2016
Format du livrable : un fichier compressé nommé *DL_ tp5_ prénom_ nom.zip* contenant le code et le résumé
Dépôt : à l'adresse `gaetan.marceau-caron@inria.fr` avec comme objet du message *DL_ tp5_ prénom_ nom.*
Description :
Le livrable associé au TP5 doit contenir le code complété et accompagné d'un résumé de 2 pages maximum. Le code doit s'exécuter avec la commande `python model.py` et afficher l'évolution de l'apprentissage (sortie par défaut du programme). Vous devez décrire l'impact de chaque paramètre du programme sur l'apprentissage (et les phrases générées automatiquement) et aussi proposer une façon d'améliorer le modèle du langage.