Université de Cergy-Pontoise,
UFR Sciences & Techniques,
Département des sciences informatiques,
Master 1 I&ISC

Intelligence Artificielle

TD 4 : Réseaux récurrents

Louis Annabi

27 avril 2021

Le but de ce TP est d'implémenter et entraîner un réseau de neurones récurrent (RNN) sur une tache de classification de documents.

1 Rappel du cours

1.1 Rétro-propagation

La rétro-propagation (backpropagation ou backprop en anglais) est l'algorithme d'optimisation dominant en deep learning. Il permet d'optimiser l'ensemble des paramètres d'un réseau de neurones en propageant le gradient par rapport à une erreur, en général calculée au niveau d'une couche de sortie (même si on peut ajouter à cette erreur des quantité dépendant des activations des couches intermédiaires ou directement des paramètres).

Dans ce TP, il n'est pas nécessaire d'implémenter l'algorithme de backpropagation. Nous utiliserons une bibliothèque (`pytorch`) s'occupant de ces calculs, nous avons juste à définir le réseau de neurones à entraîner.

1.2 Réseau récurrent

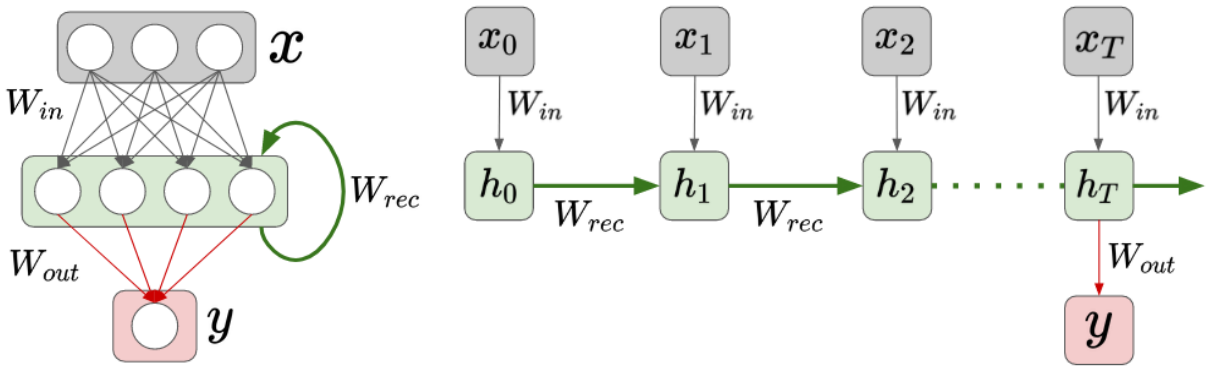

FIGURE 1 – Réseau récurrent. Gauche : Schéma du réseau de neurones. Droite : Déroulement temporel du réseau récurrent.

Un RNN est en général composé d'une couche d'entrée, d'une couche récurrente, et d'une couche de sortie. La particularité du RNN provient du fait qu'il possède une couche cachée dont l'activité évolue dans le temps. Ainsi, l'activation de la couche h_t au temps t va dépendre de sa valeur passée h_{t-1}, ainsi que des entrées à l'instant t, x_t.

Dans ce TP, nous allons implémenter le RNN défini par les équations suivantes :

$$h_t = \text{sigmoid}(W_{rec} \cdot h_{t-1} + W_{in} \cdot x_t) \tag{1}$$

$$y_T = \text{sigmoid}(W_{out} \cdot h_T) \tag{2}$$

2 Travail demandé

Pour ce TP, vous aurez besoin d'installer les bibliothèques `pytorch` et `gensim` sur python.

Nous allons construire et entraîner un RNN sur une tâche de classification de textes. La base de données contient 10000 reviews de films étant classifiées comme "positives" ou "negatives".

Chaque texte est une séquence de mots $(w_1, \ldots, w_T)$ de longueur variable. Nous allons utiliser une couche de représentation (word embedding) pré-entraînée pour obtenir une entrée $(x_1, \ldots, x_T)$ où chaque x_i correspond à un encodage sémantique du mot w_i dans un espace de dimension 50.

Nous allons entraîner notre réseau récurrent à classifier chaque texte en prédisant s'il s'agit d'une review positive $y = 1$ ou négative $y = 0$. Nous séparerons notre dataset en 9000 textes pour l'apprentissage et 1000 textes pour la validation.

2.1 Implémentations

— Implémentez le modèle de RNN décrit par les équations précédentes.
— Implémentez la boucle d'apprentissage de votre modèle sur la base de données d'entraînement.
— Implémentez une fonction permettant de classifier une entrée.

2.2 Analyses

— Entraînez le modèle avec une dimension de $d = 5$ pour la couche cachée. Une bonne manière de décider quand arrêter l'entraînement est de calculer à la fin de chaque itération la précision sur les données de validation. Quand cette précision ne diminue plus, on arrête l'apprentissage.
— Comparez la précision de votre classifieur sur les données d'apprentissage et de validation. Commentez le résultat.

2.3 Questions

— Si vous deviez utiliser un RNN pour apprendre à générer une séquence $(y_0, \ldots, y_T)$ en fonction d'une entrée $x_{t=0}$, quelle architecture de RNN utiliseriez vous ? (faire un schéma similaire au schéma de droite figure 1)
— Quel est le nombre de paramètres dans le réseau en fonction de la dimension d de la couche cachée du RNN ?
— Quel sont les intérêts d'utiliser une représentation sémantique des mots ?
— Comment pourriez-vous améliorer la précision de votre modèle ?