AS - TP 6
Réseaux récurrents : LTSM, GRU, et autres cellules à mémoire

Nicolas Baskiotis - Benjamin Piwowarski

2019-2020

Introduction (brève, cf cours)

Les réseaux récurrents (RNN) sont un type d'architectures privilégié pour traiter les séquences en "encodant" de manière itérative une séquence (cf. TP 4), par le biais d'une fonction f calcule l'état suivant par une étape d'encodage : $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$.

Les RNNs ont du mal à capturer les dépendances à long terme. Ceci est du au fait que le gradient est très instable lorsqu'on remonte à quelques pas de temps ; en simplifiant à l'extrême : la contribution de l'entrée t à pour l'étape $t+k$ sera alors de l'ordre de (cf [2])

$$\|\mathbf{A}^k\|^2 = \sum |\lambda_i|^{2k}$$

Dans ce TP, nous allons étudier des variantes des RNNs, et en particulier les Long-Short Term Memories/LSTMs [3] et les Gated Recurrent Units/GRU [1].

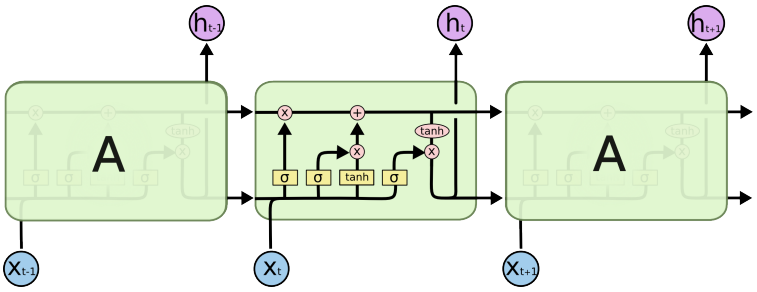

FIGURE 1 – Un LSTM (Cola'h blog)

En particulier, les LSTMs sont définis par un état externe h_t (analogue à ceux des RNNs usuels) et un état interne C_t qui représente la mémoire "à long terme" du réseau. À chaque pas de temps, C_t est mis-à-jour en fonction de de l'état interne précédent h_{t-1} et de l'entrée x_t en spécifiant ce qui doit être "oublié" du passé et ce qui doit être "retenu" pour le futur. Le nouveau état externe est calculé à partir du nouveau état interne. Ce mécanisme d'écriture repose sur la notion de *porte* empruntée à la logique, qui permet de masquer une partie du signal qui a peu d'intérêt. Dans le cas des réseaux de neurones, une porte est une fonction continue (et non discrète comme en logique), souvent une couche linéaire suivie d'une activation sigmoïde (qui produit donc une sortie entre 0 et 1).

L'évolution des états internes et externes est définie par les équations suivantes (Figure 1) qui repose sur trois portes (oubli, entrée, et sortie) :

$$
\begin{array}{rcll}
f_t & = & \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) & \text{Porte (oubli)} \\
i_t & = & \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) & \text{Porte (entrée)} \\
C_t & = & f_t \otimes C_{t-1} + i_t \otimes \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) & \text{Mise à jour (mém. interne)} \\
o_t & = & \sigma\left(W_o\,[h_{t-1}, x_t] + b_o\right) & \text{Porte (sortie)} \\
h_t & = & o_t \otimes \tanh\left(C_t\right) & \text{Sortie}
\end{array}
$$

où $[a, b]$ est une concaténation vectorielle et $\otimes$ est un produit terme à terme (Hadamard).

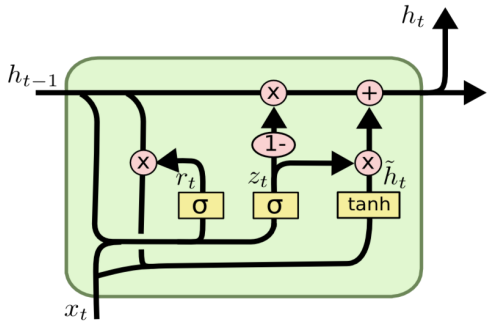

FIGURE 2 – Un GRU (Cola'h blog)

Les GRUs sont des simplifications des LSTMs (il en existe d'autres) – l'état interne et externe est le même, les portes entrée/oubli sont fusionnées (avec $i_t := 1 - f_t$) ; les équations qui définissent un GRU (Figure 2) sont :

$$
\begin{array}{rcl}
z_t & = & \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \\
r_t & = & \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \\
h_t & = & (1 - z_t) \otimes h_{t-1} + z_t \otimes \tanh\left(W \cdot [r_t \otimes h_{t-1}, x_t]\right)
\end{array}
$$

Exercice : Génération de séquences

Reprendre l'exercice 3 du TP 4 (génération de séquences), ainsi que le jeu de données (discours pré-électoraux de Trump). Vous comparerez LSTM, GRU et RNN simples sur cette tâche.

Par rapport au TP 4 :

— chaque séquence sera une phrase (qui peut se terminer par un point, un point d'interrogation ou un point d'exclamation). Nous ne supposons donc plus que les séquences ont une taille fixe. Il faut dès lors :
  — employer un symbole spécial qui marque la fin d'un séquence (EOS). Lors de l'apprentissage, il faut ajouter ce token à chaque séquence ;
  — padder chaque séquence, i.e. ajouter le caractère nul afin que les séquences d'un même batch est toutes la même longueur (cf TP 5) ;

  De plus, il faut prendre soin de ne pas inclure le padding lorsqu'on calcule le critère de maximum de vraisemblance : pour cela, la solution la plus courante (jusqu'à l'apparition des `packedsequence` que vous n'utiliserez pas dans ce TP) est d'utiliser un masque binaire (0 lorsque le caractère est nul) qui est multiplié avec les log-probabilités avant de les additionner (le paramètre `reduce=None` dans une fonction de coût - en particulier pour la cross entropie - permet d'obtenir le coût pour chaque élément plutôt que la moyenne) ;

— vous utiliserez des *embeddings* directement (cf. TP 5) par l'intermédiaire du module `nn.Embeddings`, et ne passerez donc plus par une représentation *one-hot*.

Extensions

— (obligatoire) Utiliser tensorboard pour surveiller la magnitude des gradients ainsi que l'évolution des valeurs des différentes portes. Utiliser pour cela `add_histogram`. Si vous avez des problèmes de stabilité et que les magnitudes des gradients sont grandes, utilisez du "gradient clipping".

— (obligatoire) Lors de vos tentatives de génération, vous observerez que en prenant à chaque pas de temps l'argmax vous obtiendrez très peu souvent des phrases intelligibles (vous faîtes en fait une approximation gloutonne du maximum de vraisemblance). Dans un premier temps, vous pouvez échantillonner à chaque pas de temps dans la distribution inférée, mais le résultat ne sera pas bien meilleur. La solution usuelle consiste à utiliser un "beam search" pour approximer l'argmax sur toute la séquence engendrée : le beam search consiste à conserver à tout moment t un ensemble de K séquences (et leur log-probabilité associée) ; à l'étape $t+1$, on génère pour chacune des séquences s les K symboles les plus probables étant donnée s. Puis on sélectionne les K séquence de taille $t+1$ les plus probables (et on ré-itère).

— Utiliser une segmentation de type SentencePiece (cf TP 5)

Références

— Illustrated Guide to LSTM's and GRU's: A step by step explanation

— Cola'h blog (LSTM)

Références

[1] Kyunghyun Cho et al. "Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation". In : 2014.

[2] Sepp Hochreiter et al. "Gradient Flow in Recurrent Nets : The Difficulty of Learning Long-Term Dependencies". In : (2001). URL : `http://www.bioinf.jku.at/publications/older/ch7.pdf` (visité le 09/10/2017).

[3] Sepp Hochreiter et Jürgen Schmidhuber. "Long Short-Term Memory". In : 2006.