UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

TD 3 — RNN

Exercice 1 : RNN "Elman"

On considère un réseau de neurones récurrent de type "Elman" (Elman, 1990) qui est régi par les équations suivantes :

$$h_t = \tanh(Ux_t + Wh_{t-1})$$
$$y_t = \text{softmax}(Vh_t)$$

Avec U, V, W des matrices entraînables. h_t et y_t sont respectivement l'état caché de la cellule RNN au temps t et la sortie de la cellule au temps t.

Question 1.

— Dessiner une représentation "déroulée" de ce réseau pour une séquence d'entrée de taille 3 : $\{x_0, x_1, x_2\}$

Question 2.

— Donner l'expression de y_2 en fonction de x_2 et h_1
— Puis développer l'expression pour obtenir l'expression en fonction de x_0, x_1, x_2

Remarque : on suppose que $h_{-1} = 0$.

Exercice 2 : implantation Pytorch d'un RNN "à la main"

Pytorch met à disposition une classe RNN qui permet d'instancier un RNN de type Elman multicouche [1]. Cependant, dans cet exercice nous souhaitons implanter nous-mêmes un

1. voir `https://pytorch.org/docs/stable/generated/torch.nn.RNN.html`

RNN Elman.
Considérons un problème de classification de séquences. Par exemple, nous souhaitons identifier la langue d'une séquence de caractères de longueur donnée T (par ex. $T = 10$ lettres), parmi n langues possibles (par ex. $n = 3$ avec fr, eng, pt).
on veut un modèle régi par les équations suivantes :

$$h_t = \text{ReLU}(Ux_t + Wh_{t-1} + b_h)$$
$$y = \text{softmax}(Vh_T + b_y)$$

Avec U, V, W, b_h, b_y des paramètres entraînables. h_t et y sont respectivement l'état caché de la cellule RNN au temps t et la sortie de la cellule au temps final T.

Question 1.

— Compléter le code suivant pour créer ce modèle :

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class myRNN(nn.Module):

  def __init__(self, ??):
    super().__init__()
    ??

  def forward(self, ??):
    ??
```

Question 2.

— Compléter les lignes suivantes pour tester ce modèle, quelle est la dimension de la sortie out ?

```
B=5
T=13
d=7
nb_class=3
model = myRNN(??)
input = torch.randn(??)
out = ??
print(out.size())
```

Remarque : ici nous avons traité des séquences de taille identique. L'un des intérêts des RNN est justement de pouvoir traiter des séquences de longueur variable. En Pytorch,

l'une des façons de faire est d'empaqueter ces séquences à l'aide de la fonction `torch.nn.utils.rnn.pack_padded_sequence()`. Elle prend en entrée un tenseur contenant les séquences "paddées" (on a ajouté des zéros ou un autre symbole pour que toutes les séquences aient la même taille), ainsi que la liste des longueurs de chacune des séquences. L'objet RNN de Pytorch accepte ces tenseurs qui sont des objets `PackedSequence`. L'avantage est que les calculs ne sont faits que sur les éléments des séquences et pas sur les zéros ajoutés pour le padding.

Exercice 3 : Backprop Through Time (BPTT)

On reprend le réseau de neurones de l'exercice 1.

Questions

- Calculer le gradient $\frac{\partial h_0}{\partial U}$ à l'aide d'un graphe de calcul et le mode reverse de la différentiation automatique
- Puis faire de même pour $\frac{\partial h_1}{\partial U}$
- En déduire par généralisation $\frac{\partial h_2}{\partial U}$

Rappel : $\tanh' = 1 - \tanh^2$