AS - TP 4
Réseaux récurrents

Nicolas Baskiotis - Benjamin Piwowarski

2019-2020

Introduction (brève, cf cours)

Les réseaux récurrents (RNN) sont un type d'architectures privilégié pour traiter les séquences. De façon générique, un réseau récurrent utilise un état caché (ou *mémoire*) pour accumuler l'information des itérations précédentes. Lors du calcul à un moment t de la séquence, le réseau utilise à la fois l'entrée à l'instant t et l'état caché pour inférer un nouvel état caché. L'état caché peut être décodé pour prédire une valeur de sortie correspondant à l'instant t mais également être utilisé pour continuer l'inférence au temps suivant.

Formellement, un réseau récurrent f calcule l'état suivant par une étape d'encodage : $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$. Un exemple simple de réseau est $f(\mathbf{x}_t, \mathbf{h}_t) = tanh(\mathbf{W}_i\mathbf{x}_t + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b}_h)$ qui opère une transformation linéaire sur les entrées, une sur l'état caché, combine les deux puis applique une non-linéarité (tangente hyperbolique, sigmoïde ou ReLU par exemple). L'état caché peut être décodé également avec un réseau linéaire combiné à une non-linéarité : $\mathbf{y}_t = d(h_t) = \sigma(\mathbf{W}_d\mathbf{h}_t + \mathbf{b}_d)$.

Par exemple, soit $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$ une séquence en entrée (possiblement multivariée : $\mathbf{x}_i \in \mathbb{R}^d$) et $\mathbf{h}_0$ un état caché initial. Pour calculer le décodage de cette séquence, les étapes sont les suivantes :

$$\mathbf{h}_1 = f(\mathbf{x}_1, \mathbf{h}_0),\ \mathbf{y}_1 = d(\mathbf{h}_1),\ \mathbf{h}_2 = f(\mathbf{x}_2, \mathbf{h}_1),\ \mathbf{y}_2 = d(\mathbf{h}_2), \ldots,\ \mathbf{h}_T = f(\mathbf{x}_T, \mathbf{h}_{T-1}),\ \mathbf{y}_T = d(\mathbf{h}_T)$$

Il existe de multiples possibilités pour inférer un état caché. Une première variante simple est d'avoir à la place de deux réseaux linéaires séparés pour les entrées et l'état caché un réseau commun : on opère une concaténation des entrées et de l'état caché et on utilise une matrice de poids unique sur ce vecteur concaténé. Nous verrons dans les TPs suivants des architectures plus complexes.

À partir de ce principe général, les RNNs sont très flexibles et de multiples applications existent :

— l'état caché peut être décodé uniquement à la fin d'une séquence pour produire une sortie unique. Ce type de réseau, appelé *many-to-one*, permet par exemple de classifier une séquence : le décodage de la sortie indique alors la classe de la séquence ;

— l'état caché peut être décodé à chaque pas de temps (type *many-to-many*), pour du forecasting (prédiction de températures, d'embouteillages) ou de la génération de texte ;

— une variante consiste à lire toute une séquence puis produire une séquence à partir de l'état caché final (en particulier pour la traduction) ;

— il peut y avoir également qu'une seule entrée à partir de laquelle on produit une séquence (type *one-to-many*).

Dans la suite de ce TP, nous étudierons successivement le problème de la classification de séquences, du forecasting et enfin de la génération.

Exo 1 : Classification de séquences

Télécharger le jeu de données des températures. Chaque ligne représente la température chaque heure pour 30 villes américaines.

L'objectif est de construire un modèle qui à partir d'une séquence de température de longueur non prédéfinie infère la ville correspondante. Typiquement, ce genre de problème est impossible à traiter avec un réseau classique : même en considérant des séquences de longueur fixée, les dimensions ne sont pas homogènes en fonction des séquences (une dimension ne représente pas toujours la même heure).

Dans toute la suite, nous considérerons que nos modèles peuvent prendre en entrée un batch de séquences. La dimension de l'entrée sera toujours $length \times batch \times dim$ **avec** $length$ **la longueur des séquences,** $batch$ **la taille du batch et** dim **la dimension d'un élément de la séquence.**

Coder une classe `RNN` qui implémente un réseau récurrent simple tel que décrit en introduction. La méthode `forward(x,h)` doit traiter toute la séquence passée en paramètre. Il est donc conseillé de faire une fonction `one_step(x,h)` qui permet de traiter un pas de temps de la séquence et que `forward` appelle successivement sur toute la séquence. Par ailleurs, le décodage peut se faire soit dans la méthode `forward` soit dans une autre fonction.

On aura donc pour un réseau de dimension latente *latent* les entrées/sorties suivantes :

— `forward(x,h)` avec `x` de dimension $length \times batch \times dim$, `h` de dimension $batch \times latent$, qui retourne un tenseur de dimension $length \times batch \times latent$ (la succession des états cachés) ;

— `one_step(x,h)` avec `x` de dimension $batch \times dim$ et `h` de dimension $batch \times latent$ qui renvoie un tenseur de dimension $batch \times latent$ et potentiellement un tenseur de dimension $batch \times dimout$.

Pour la classification multi-classe, il est usuel d'utiliser une couche linéaire, suivie d'un softmax couplé à un coût de cross-entropie.

Tester votre implémentation sur le jeu de données, en sélectionnant un sous-ensemble d'une dizaine de villes. Vous pouvez considérer dans cette partie que pour l'apprentissage du modèle, les séquences des batchs ont toutes la même longueur. Attention toutefois à tirer aléatoirement le début des séquences d'apprentissage (afin de ne pas avoir un jeu de données biaisé qui commence toujours aux mêmes heures). Par ailleurs, vous pouvez utiliser un tenseur nul pour l'état initial. Observer le coût en apprentissage et en test, ainsi que le taux d'erreurs (coût 0-1). Faites varier pour le test la longueur des séquences.

Exo 2 : Modèle de séquences multi-variées

L'objectif de cette partie est de faire de la prédiction de séries temporelles : à partir d'une séquence de température de longueur t pour l'ensemble des villes du jeu de données, on veut prédire la température à $t+1$, $t+2$, ...

Que doit-on changer au modèle précédent ? Quel coût est dans ce cas plus adapté que la cross-entropie ?

Variantes :

— pour quantifier à quel point la corrélation entre les séries temporelles est prise en compte, vous entrainerez deux types de modèle : (1) en prédisant n séries temporelles dans $\mathbb{R}$ avec n ensemble de paramètres ; et (2) en prédisant une série temporelle dans $\mathbb{R}^n$.

— Faire les expériences en faisant varier l'horizon de prédiction (à $t+2$, etc.) et la longueur des séquences en entrée.

Exo 3 : Génération de séquences

Principe La génération de séquences consiste à produire une séquence de symboles discrets à partir d'une séquence en entrée. L'objectif est donc de décoder à partir de l'état caché une distribution de probabilités multinomiale sur les symboles à engendrer. Il faut donc une dimension de sortie du nombre de symboles considéré et utiliser un softmax pour obtenir une distribution à partir du décodage de l'état caché. Le coût cross-entropie est adapté pour apprendre cette distribution.

Embedding Lorsque la séquence d'entrée est également discrète, il faut tout d'abord projeter cet espace d'entrée dans un espace continu - ce qu'on appelle un *embedding*. Supposons que l'on ait n symboles à encoder. Une première étape est de faire un encodage appelé *one-hot* dans $\mathbb{R}^n$: chaque symbole se voit attribuer un

index (on obtient ainsi un dictionnaire des symboles) et est représenté dans un espace continu par un vecteur de dimension n nul partout sauf dans la dimension de son index mise à 1. La deuxième étape consiste à apprendre une représentation dans un deuxième espace de dimension n' (avec $n' < n$ en général - pour le cas des caractères, il n'y aura pas une grande différence) par une projection linéaire. Cet apprentissage peut se faire indépendamment de la tâche de génération ou plus classiquement en même temps que l'apprentissage du réseau général (ce que vous ferez pendant ce TP). Cette représentation peut être ensuite utilisée comme entrée du réseau.

Génération Une fois le réseau appris, la génération se fait (soit à partir d'un début de séquence, soit à partir d'un état initial vide) en choisissant le symbole le plus probable dans la distribution multinomiale décodée à chaque pas de temps. Ce symbole est ensuite considéré comme entrée au pas de temps suivant et la génération se poursuit itérativement. Une autre possibilité est d'échantillonner suivant la multinomiale pour obtenir plusieurs échantillons.

Dans ce TP, nous nous limiterons à la génération de séquences de **taille fixe** que l'on déterminera à l'avance.

Expériences Le deuxième jeu de données fourni est un ensemble de discours pré-électoraux de Trump. Le but est d'apprendre un RNN qui permet de engendrer un discours à la Trump.

Vous pouvez utiliser le bout de code suivant pour pré-traiter le texte (enlève les caractères accentués et non usuels) et transformer en liste d'index une chaîne de caractères :

```
LETTRES = string.ascii_letters + string.punctuation+string.digits+' '
id2lettre = dict(zip(range(1,len(LETTRES)+1),LETTRES))
id2lettre[0]='' ##NULL CHARACTER
lettre2id = dict(zip(id2lettre.values(),id2lettre.keys()))

def normalize(s):
    return ''.join(c for c in unicodedata.normalize('NFD', s) if  c in LETTRES)
def string2code(s):
    return torch.tensor([lettre2id[c] for c in normalize(s)])
def code2string(t):
    if type(t) !=list:
        t = t.tolist()
    return ''.join(id2lettre[i] for i in t)
```