Université Mohammed V
Faculté des Sciences
Rabat

L'apprentissage profond pour le texte et les séquences

Mehdi Bouskri

mehdi_bouskri@um5.ac.ma

Introduction

L'apprentissage profond pour le traitement du texte est simplement la reconnaissance des formes appliquée aux mots, aux phrases et aux paragraphes, de la même manière que la vision par ordinateur est simplement la reconnaissance des formes appliquée aux pixels.

L’utilisation de données textuelles

les modèles d'apprentissage profond ne prennent pas en entrée de texte brut donc la vectorisation des données est une étape nécessaire qu’on peut la faire de plusieurs façons:

- En segmentant le texte en mots, et en transformant chaque mot en vecteur.
- En segmentant le texte en caractères, et en transformant chaque caractère en un vecteur.
- En extrayant des "N-grams" de mots, et en transformant chaque N-gram en un vecteur.

L’utilisation de données textuelles

De manière générale, les différentes unités dans lesquelles vous pouvez décomposer le texte (mots, caractères ou N-grammes) sont appelées "tokens", et la décomposition du texte en tokens est appelée "tokenization".

Tous les processus de vectorisation de texte consistent à appliquer un certain schéma de tokenization, puis à associer des vecteurs numériques aux tokens générés.

One-hot encoding

One-hot encoding est la manière la plus courante et la plus basique pour transformer un token en un vecteur.

Elle consiste à associer un indice entier unique à chaque token, puis à transformer cet indice entier i en un vecteur binaire de taille N, le nombre de tokens uniques, ce vecteur serait entièrement nul à l'exception de la i-ième entrée, qui serait 1.

One-hot hashing

Une variante de one-hot encoding qui peut être utilisée lorsque le nombre de tokens uniques dans votre vocabulaire est trop important pour être traité de manière explicite.
Au lieu d'attribuer explicitement un indice à chaque token et de conserver une référence de ces indices dans un dictionnaire, on peut hacher les tokens dans des vecteurs de taille fixe.

Word embeddings

Contrairement aux vecteurs obtenus par one-hot encoding qui sont binaires, dispersés et de haute dimension, les "word embeddings" sont des vecteurs de nombres réels, de petite dimension, qui sont appris à partir des données.

Word embeddings

Il y a deux façons d'obtenir des word embeddings:

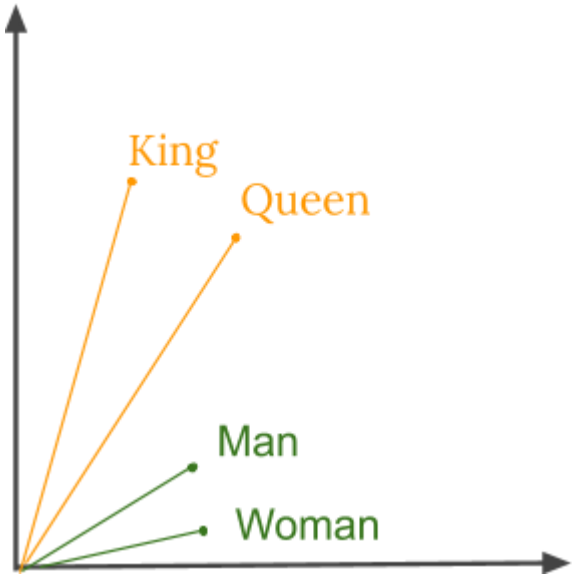
- Apprendre les word embeddings conjointement avec la tâche principale qui vous intéresse.

- Importez dans votre modèle des word embeddings qui ont été précalculés en utilisant une tâche d'apprentissage différente.

Apprentissage des word embeddings avec la couche Embedding

La façon la plus simple d'associer un vecteur dense à un mot serait de choisir le vecteur au hasard. Le problème avec cette approche est que l'espace des word embeddings résultant n'aurait pas de structure.

Dans les espaces de word embeddings du monde réel, les exemples courants de transformations géométriques significatives sont les "vecteurs de genre" et les "vecteurs de pluriel"



Apprentissage des word embeddings avec la couche Embedding

Il est raisonnable de créer un nouvel espace d'intégration pour chaque tâche spécifique. Il s'agit juste d'apprendre les poids de la couche Embedding.

```
from keras.layers import Embedding
embedding_layer = Embedding(input_dim, 64)
```

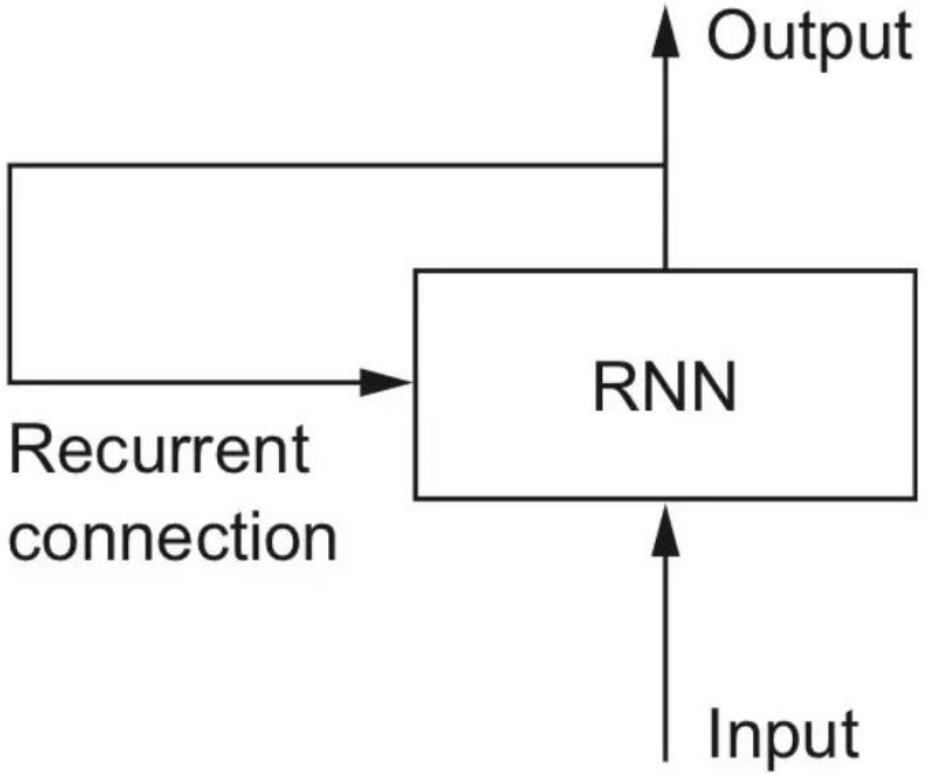
Cette couche prend au moins deux arguments : `input_dim` le nombre de tokens possibles, et `output_dim` la dimensionnalité des embeddings.

Utilisation des word embeddings pré-entraînés

Au lieu d'apprendre les embeddings de mots conjointement avec le problème que vous voulez résoudre, on peut charger des vecteurs d'embeddings à partir d'un espace d'embeddings précalculé connu pour être hautement structuré et présentant des propriétés utiles qui capture des aspects génériques de la structure du langage.

Réseaux de neurones récurrents

Les RNNs traitent les séquences en itérant à travers les éléments de la séquence et en maintenant un état contenant des informations relatives à ce qu'ils ont vu jusqu'à présent.



Pseudo-code d'un RNN simple

```
state_t = 0
for input_t in input_sequence:
    output_t = f(input_t, state_t)
    state_t = output_t
```

```
state_t = 0
for input_t in input_sequence:
    output_t = activation(dot(W, input_t) + dot(U,
state_t) + b)
    state_t = output_t
```

La couche SimpleRNN de Keras

SimpleRNN traite des lots de séquences, comme toutes les autres couches Keras. Cela signifie qu'il prend des entrées de forme (batch_size, timesteps, input_features).

Elle peut être exécutée dans deux modes différents :

- retourner les séquences complètes de sorties successives pour chaque timestep.
- retourner seulement la dernière sortie pour chaque séquence d'entrée.

Ces deux modes sont contrôlés par l'argument return_sequences

les couches LSTM et GRU

Considérons la couche LSTM ("Long-Short Term Memory"), il s'agit d'une variante du RNN simple, qui ajoute un moyen de transmettre des informations sur plusieurs timesteps.

Pour comprendre le fonctionnement des LSTM, ajoutons au simpleRNN un flux de données supplémentaire qui transmet des informations entre les timesteps, C.

```
output_t = activation(dot(state_t, Uo) +
    dot(input_t, Wo) + dot(C_t, Vo) + bo)
```

les couches LSTM et GRU

On obtient le nouvel état de transfert C_t en combinant simplement trois transformations distinctes qui ont la forme d'un simple RNN:

```
c_t+1 = i_t * k_t + c_t * f_t
```

Avec:

```
i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)
```

les couches LSTM et GRU

La couche GRU ("Gated Recurrent Unit") ne possède pas de mémoire interne et elle a deux transformations (ou porte):

- La porte de mise à jour est chargée de déterminer la quantité d'informations précédentes qui doivent être transmises à l'état suivant.

- La porte de réinitialisation est utilisée par le modèle pour déterminer la quantité d'informations antérieures à négliger.

Caractéristiques avancées des RNN

Nous allons aborder trois techniques pour améliorer les performances et le pouvoir de généralisation des réseaux de neurones récurrents:

- Le dropout récurrent, une manière spécifique d'utilisation du dropout pour réduire le surajustement dans les couches récurrentes.
- La superposition de couches récurrentes, pour augmenter le pouvoir de représentation du réseau.
- Les couches récurrentes bidirectionnelles, qui présentent la même information à un réseau récurrent de différentes manières

Utilisation du dropout récurrent

La manière correcte d'utiliser le dropout avec un réseau récurrent est d'appliquer le même masque de dropout (le même modèle d'unités abandonnées) à chaque timestep, au lieu d'un masque de dropout qui peut varier aléatoirement d'un timestep à l'autre.

En plus, afin de régulariser les représentations formées par les portes récurrentes de couches telles que GRU et LSTM, un masque de dropout constant dans le temps doit être appliqué aux activations récurrentes internes de la couche.

Utilisation du dropout récurrent

Chaque couche récurrente dans Keras possède deux arguments liés au dropout :

- `dropout`, spécifiant le taux de dropout pour les unités d'entrée de la couche.
- `recurrent_dropout`, spécifiant le taux de dropout des unités récurrentes

Superposition de couches récurrentes

Pour superposer des couches récurrentes dans Keras, toutes les couches intermédiaires doivent retourner leur séquence complète de sorties plutôt que leur sortie du dernier timestep.

Utilisation de RNN bidirectionnel

Un RNN bidirectionnel est une variante courante des RNN qui peut offrir de meilleures performances qu'un RNN ordinaire pour certaines tâches.

Il s'agit simplement de deux RNN réguliers, tels que les couches GRU ou LSTM, qui traitent chacun la séquence d'entrée dans une direction (chronologique et antichronologique), puis fusionnent leurs représentations.

Amélioration des performances

Vous pouvez essayer d'autres techniques afin d'améliorer la performance du modèle:

- Ajuster le nombre d'unités dans chaque couche récurrente dans la configuration superposée.
- Ajuster le taux d'apprentissage.
- Utiliser des couches LSTM au lieu des couches GRU.
- Utiliser un plus grand réseau dense au-dessus des couches récurrentes.

Traitement des séquences avec des convnets

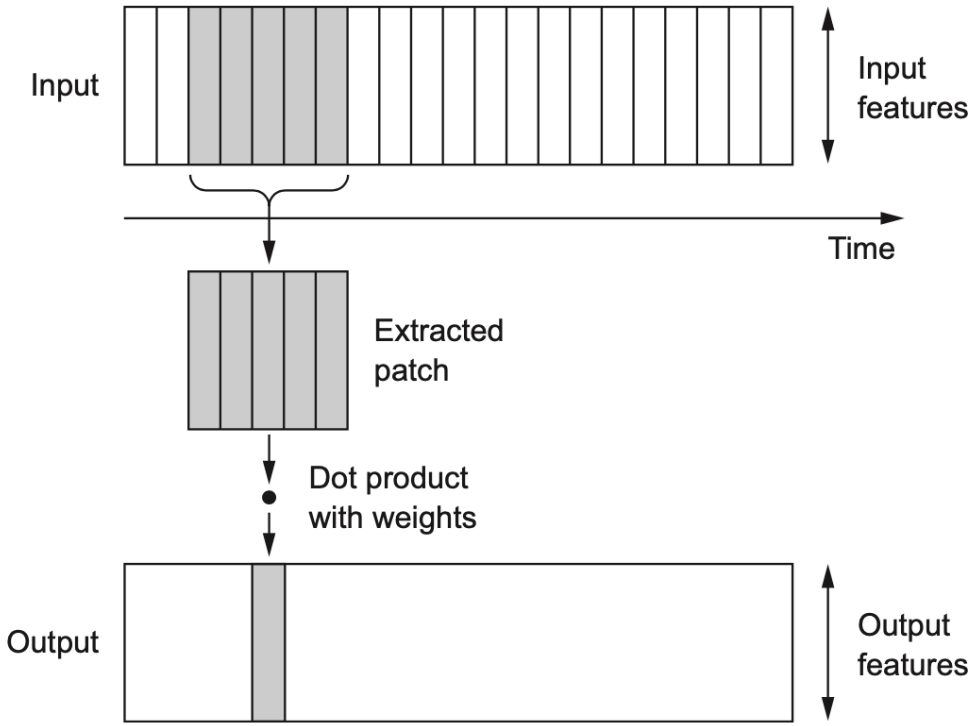
Les propriétés qui rendent les convnets excellents en vision par ordinateur sont également pertinentes pour le traitement des séquences. En effet, le temps peut être traité comme une dimension spatiale, comme la hauteur ou la largeur d'une image 2D.

Convnets 1D comme alternative aux RNN pour le traitement des séquences

Les convnets 1D peuvent être compétitifs avec les RNN sur certains problèmes de traitement de séquences, avec un coût de calcul généralement beaucoup moins élevé. Les petits convnets 1D peuvent offrir un alternatif rapide aux RNN pour des tâches simples telles que la classification de textes ou la prédiction de séries temporelles.

Convnets 1D comme alternative aux RNN pour le traitement des séquences

On peut utiliser des convolutions 1D comme outil pour extraire des motifs 1D locaux (sous-séquences) à partir de séquences.



1D Pooling pour les données séquentielles

L'opération de pooling 2D a un équivalent 1D, en extrayant des segments 1D (sous-séquences) d'une entrée et en sortant la valeur maximale ("max pooling") ou la valeur moyenne ("average pooling").

De même que dans les convnets 2D, il est utilisé pour réduire la longueur des entrées 1D.

Implémentation d'un convnet 1D

Les convnets 1D sont structurés de la même manière que leurs homologues 2D, ils consistent en une superposition de couches Conv1D et MaxPooling1D.

La couche Conv1D prend en entrée des tenseurs 3D avec une forme (échantillons, temps, caractéristiques) et retourne également des tenseurs 3D de forme similaire. La fenêtre de convolution est une fenêtre 1D sur l'axe temporel, l'axe 1.

Combinaison de CNN et de RNN pour le traitement de longues séquences

Pour reconnaître des motifs longs, on pourrait superposer de nombreuses couches de convolution et de pooling, ce qui donnerait des couches supérieures qui "percevront" de longs morceaux des entrées originales, mais cela reste un moyen assez faible pour induire une sensibilité à l'ordre.

Une façon de mettre en évidence cette faiblesse est d'essayer les convnets 1D sur le problème de prédiction de température de la section précédente, où la sensibilité de l'ordre était essentielle pour produire de bonnes prédictions.

Combinaison de CNN et de RNN pour le traitement de longues séquences

Une stratégie permettant de combiner la performance des convnets avec la sensibilité à l'ordre des RNN consiste à utiliser un convnet 1D comme étape de prétraitement avant un RNN. Cette méthode est particulièrement avantageuse lorsqu'il s'agit de séquences si longues qu'elles ne pourraient pas être traitées par des RNN.

Combinaison de CNN et de RNN

Remarques importantes à retenir:

- Les convnets 1D sont performants pour le traitement des motifs. Ils offrent un alternatif plus rapide aux RNN pour certains problèmes.
- En général, les convnets 1D sont structurés comme leurs équivalents 2D , ils consistent en des superpositions de couches Conv1D et de couches Pooling1D, et terminant finalement par une opération globale de pooling ou d'aplatissement.
- Puisque les RNN sont très coûteux pour traiter de longues séquences, par rapport aux convnets 1D, il peut être utile d'utiliser un convnet 1D comme étape de prétraitement avant un RNN, en réduisant la séquence et en extrayant des représentations utiles pour être traitées par le RNN