Projet : analyse syntaxique

1 Problématique

Le projet consiste à réaliser un analyseur syntaxique statistique en dépendances qui sera entrainé et évalué sur le corpus Sequoia. Le projet comporte deux sous-questions:

- Réaliser un tagger capable d'assigner des parties du discours aux mots.
- Réaliser un analyseur capable de prédire une analyse en dépendances projective (non nécessairement typée) sur des données de test.

Le tagger sera nécessairement un tagger neuronal de type RNN. La ou les représentations cachées (mémoire) du tagger seront utilisées comme features par le système de pondération du parser.

2 Découpage du corpus

On attend en préambule que vous produisiez des méthodes de lecture du corpus et de découpage de celui-ci en trois sous-parties: train, dev et test dans un fichier appelé `corpus.py`.

3 Réalisation du tagger

Le tagger sera nécessairement un tagger de la famille RNN (LSTM ou GRU) et sera implémenté avec la librairie Keras ***en utilisant l'API dite fonctionelle***. En supposant que chaque mot w_t d'une phrase $w_1 \dots w_n$ est codé par un vecteur one-hot $\mathbf{x}_t$ ($\mathbf{x}_t = \text{onehot}(w_t)$), le tagger implémente (a minima) la structure de réseau suivante:

$$\begin{aligned}
\mathbf{y}_t &= \text{softmax}(\mathbf{W}_0\mathbf{h}_t) && (1 \leq t \leq n)\\
\mathbf{h}_t &= g(\mathbf{W}_1\mathbf{e}_t + \mathbf{W}_2\mathbf{h}_{t-1} + \mathbf{b}) && (1 \leq t \leq n)\\
\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t && (1 \leq t \leq n)
\end{aligned}$$

Vous pouvez également expérimenter avec des variantes comme par exemple BI-RNN. Le tagger sera implémenté dans une classe appelée `NNtagger` dans un fichier appelé `tagger.py`. La classe comportera au moins trois méthodes:

- `tag(self,word_list)` qui renvoie une liste de tags y_t $(1 \leq t \leq n)$ prédits pour la séquence de mots `word_list`
- `predict(self,wordlist)` qui renvoie la liste des vecteurs $\mathbf{h}_t$ pour $1 \leq t \leq n$.
- `train(corpus)` qui entraine le tagger à partir d'un corpus. Cette dernière méthode peut inclure des paramètres additionnels de votre choix.

Question 1 On attend que votre tagger gère les mots inconnus. Vous présenterez la méthode que vous aurez mise au point pour ce faire.

Question 2 On attend que vous testiez le paramétrage du tagger: taille des couches cachées, taux d'apprentissage, variantes dans le traitement des mots inconnus, etc. de manière à maximiser ses performances sur le corpus de développement. Vous présenterez la méthode que vous aurez utilisée pour ce faire et indiquerez quels paramètres vous semblent importants à régler et pourquoi.

4 Analyseur syntaxique

La seconde partie du travail consiste à implémenter un analyseur syntaxique original. dont le système de pondération prend en entrée les vecteurs de mots produits par le tagger RNN. On pourra par exemple utiliser un algorithme du perceptron structuré pour ce faire ou des algorithmes locaux comme un réseau de neurones à propagation avant.

L'analyseur sera un analyseur en dépendances (non nécessairement typées) et projectif. L'algorithme sera choisi parmi Arc-eager, Arc-standard ou Arc-factored. La méthode d'exploration des solutions sera au choix gloutonne, en faisceau ou utilisera une méthode de recherche inspirée par l'algorithme de Dijkstra.

Le parser sera implémenté dans une classe `DependencyParser` elle même localisée dans un fichier appelé `parser.py`

Question 3 Donner formellement le système de transitions (ou de déduction selon le cas) que votre parser implémente.

Question 4 Indiquez quelle méthode d'exploration des solutions a été retenue.

Question 5 Donner une formalisation du système de pondération utilisé par votre parser. On attend en particulier une formalisation du calcul du score d'une dérivation et une formalisation de la procédure de décision qui permet de renvoyer l'analyse de meilleur score.

Question 6 Identifier les hyperparamètres importants pour votre système d'analyse syntaxique et donner un résumé des tests que vous aurez réalisés sur le corpus de développement pour maximiser ses performances.

5 Rendu

On attend en rendu les fichiers de code `tagger.py` , `parser.py` et `corpus.py` ainsi qu'un `README` qui explique comment entrainer et tester l'analyseur. Les fichiers de code seront accompagnés d'un document de synthèse (1 à 5pp) qui présente votre travail et qui apporte des éléments de réponses aux 6 questions posées dans le sujet.

6 Remarques

Il est autorisé de réutiliser des fragments de code donnés sur le site du cours pour vous faciliter la tâche. On conseille de choisir soigneusement le type d'algorithme d'analyse à implémenter: certains sont plus difficiles que d'autres à mettre en oeuvre...