**Arc-eager**

Pour mettre en place notre parser nous avons utilisé arc-eager. Ce parser nous sera utile afin de créer nos exemples d’apprentissage, mais aussi pour créer notre fichier conllu de test.

Une configuration pour notre parser ressemble à c = ( B , S , A ) où B représente le buffer, S la stack , et A l’ensemble des arcs. Pour passer d’une configuration à une autres il faut appliquer une des quatre transitions possibles, citez ci-dessous, sur le dernier élément du stack et le premier élément du buffer.

* *LEFT\_ARC*: *Ajoute* à A un arcs wb → ws avec le label ls , où wb est le premier mot du buffer, ws le dernier mot du stack et ls le label du dernier mot du stack. Puis on retire ws du stack.
* *RIGHT\_ARC* : *Ajoute* à A un arcs ws → wb avec le label lb , et retire wb du buffer et l’ajoute au stack.
* *SHIFT* : Retire wb du buffer et l’ajoute au stack.
* *REDUCE* : Retire ws du stack.

Afin de déterminer quelle transition doit être effectuée pour passer d’une configuration à une autre nous avons utilisé deux oracles, un pour l’apprentissage et un pour le test. Ces deux oracles seront décrits par la suite.

Pour rajouter le ROOT nous avons pris comme configuration initiale S = [ROOT] , B=[w1, … , wn] et A vide , où wi représente un le mot ( structuré en fonction des besoins des features ) à la position i dans la phrase.

Lors de la partie d’apprentissage, nous avons pris comme heuristique1 de générer un *SHIFT* lorsque la stack est vide et que le premier mot du buffer est ROOT, afin d’être cohérent avec notre configuration initiale. La configuration finale de notre parser est telle que le buffer et la stack sont vides. Lors de la partie d’apprentissage, lorsque le buffer est vide nous avons pris comme heuristique2 de générer un *REDUCE* pour chaque mot restant dans la stack.

Afin de générer nos données d’apprentissages, pour chaque configuration nous utilisons ws et wb reformatés par rapport au features et la transition résultante de l’oracle.

|  |  |  |
| --- | --- | --- |
| **Feature 1**  S.0.POS  B.0.POS  DIST | **Feature 2**  S.0.POS  S.0.LEMMA  S.0.MORPHO  S.-1.POS  B.0.POS  B.0.LEMMA  B.0.MORPHO  B.-1.POS  B.1.POS  DIST | **Feature 3**  S.0.POS  S.0.LEMMA  S.0.MORPHO  S.-1.POS  B.0.POS  B.0.LEMMA  B.0.MORPHO  B.-2.POS  B.-1.POS  B.1.POS  DIST |

Pour générer nos données dans le cas des deux heuristiques, nous avons rajouté le mot *N..U..L..L* afin de remplacer les valeurs inexistante dans ces deux cas.

Du fait que nous ne traitons pas le problème des phrases non projectives, nous avons rajouté une valeur à la fin de nos features afin de différencier les données provenant d’une phrase projective ou non.

Lors de la partie de test, notre parser est utilisé pour écrire dans le fichier de test les arcs créer grâce à celui-ci.

**Oracle**

Durant la partie d’apprentissage nous avons utilisé comme oracle le fichier .conllu . Pour rappel, l’oracle permet de prédire une transition en fonction d’une configuration. Ayant une vision en continue sur les « HEAD » du fichier, il est alors possible prédire :

* *RIGHT* si le HEAD de wb (noté HEAD(wb) par la suite) est égal à position de ws dans la phrase.
* *LEFT* si HEAD(ws) est égal à la position de wb dans la phrase.
* *REDUCE* si tous les mots ayant la position de ws dans la phrase comme HEAD ont déjà étaient traités.
* *SHIFT* dans tous les autres cas.

Durant la partie de test nous avons utilisé comme oracle notre réseau entrainé pour prédire une transition en fonction d’une configuration.

**Tableau de résultat**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Langue  Features | Allemand | Anglais | Français | Japonais |
| 1 | 55.30 | 55.26 | 61.96 | 68.36 |
| 2 | 67.09 | 68.04 |  | 75.93 |
| 3 | 66.70 | 67.68 |  | 78.77 |
| 1-forms |  |  |  |  |

**Taux de phrase projective dans le test**

|  |  |  |  |
| --- | --- | --- | --- |
| **Allemand** | **Anglais** | **Français** | **Japonais** |
| 840 / 876 ≈ **96%** | 800 / 914 ≈ **87.5%** | 398 / 416 ≈ **96%** | 538 / 551 ≈ **98%** |