Rapport d'analyse de l'usage de l'outil de parsing syntaxique SyntaxNet avec un modèle français

2 Décembre 2016

Auteurs

Frédéric Aman, Laurent Besacier

Généralités

SyntaxNet est un parser syntaxique open-source qui repose sur la librairie TensorFlow utilisant des réseaux de neurones.
A partir d'une phrase d'entrée, il tague chaque mot (appelé aussi *token*) avec un *tag* grammatical (ou *POS tag* pour *port-of-speech tag* en anglais) qui décrit la fonction syntaxique du mot. Il détermine également les dépendances syntaxiques (c'est-à-dire des *arcs* associés à des *labels*) entre les mots de la phrase, représentés par un arbre de dépendances.
Par exemple, pour la phrase « Alice voit Bob », on obtient l'arbre de dépendances suivant :

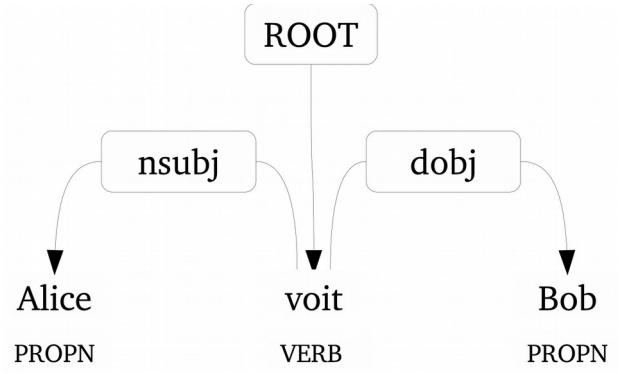

Figure 1 : Exemple d'arbre de dépendances

Alice et *Bob* sont des noms propres (*PROPN*), et *voit* est un verbe (*VERB*) qui est la racine (*ROOT*) de la phrase. *Alice* est le sujet (*nsubj*) de *voit*, tandis que *Bob* est l'objet direct (*dobj*) de *voit*.

Ainsi, SyntaxNet effectue les 4 actions suivantes :
- tokenisation : découpe la phrase en tokens en se basant sur les espaces et la ponctuation,
- POS tagging : détermine les tags grammaticaux (*POS tags*) pour chaque token,
- parsing : détermine les arcs et leurs étiquettes (*arc-labels*) de dépendances entre les tokens,
- génération de l'arbre sous forme ASCII, par exemple :

```
voit VERB ROOT
+-- Alice PROPN nsubj
+-- Bob PROPN dobj
```

SyntaxNet utilise des modèles appris sur des corpus de textes de type *treebank*, qui sont un ensemble de phrases dont les structures syntaxiques et sémantiques ont été annotées.

Un classifieur de type réseau de neurones permet de prendre les décisions sur les POS tags et les arc-labels de la phrase traitée. A chaque étape du traitement, de nombreuses décisions peuvent être possibles, le réseau de neurones donne des scores pour les décisions concurrentes en fonction de leur plausibilité. La phrase d'entrée est traitée de gauche à droite, les POS tags et arc-labels étant ajoutés progressivement à mesure que chaque mot dans la phrase est considéré.

Prédiction des POS tags (tagger)

Le tagger traite les phrases de gauche à droite ; pour le mot considéré, des *features* sont extraits de ce mot et d'une fenêtre autour de ce mot. Ils sont donnés en entrée du réseau de neurones, qui prédira la distribution de probabilités des POS tags.

3 types de features sont utilisés pour les POS tags : mots, préfixes et suffixes. Par exemple, on peut définir les features suivants :

```
 'input.token.word '
 'input(1).token.word '
 'input(2).token.word '
 'input(-1).token.word '
 'input(-2).token.word '
 'input.suffix(length="2") '
 'input(1).suffix(length="2") '
 'input(2).suffix(length="2") '
 'input(-1).suffix(length="2") '
 'input(-2).suffix(length="2") '
 'input(1).prefix(length="2") '
 'input(2).prefix(length="2") '
 'input(-1).prefix(length="2") '
 'input(-2).prefix(length="2") '
```

« input.token.word » est le mot considéré, « (length="2") » est la longueur des préfixes et suffixes, et les indices « (-2) », « (- 1) », « (1) » et « (2) » correspondent aux indices des mots de la fenêtre (2 mots avant et 2 mots après par rapport au mot considéré).

Les features sont concaténés dans une matrice, qui est fournie en entrée des couches cachées du réseau de neurones.

Prédiction des dépendances (parser)

Dès lors que le système peut prédire le rôle grammatical des mots, il lui faut aussi prédire comment les mots dans la phrase sont reliés les uns aux autres. Le réseau de neurones prend notamment les sorties du modèle de POS tag comme *features* pour le parser.

Ces dépendances syntaxiques sont de type *head-modifier* (*tête-modificateur*). Pour chaque mot considéré (appelé le *modificateur*), le système va choisir une *tête*, créant ainsi un *arc* entre la *tête* et le *modificateur*, et l'arc est associé à un rôle grammatical (le *label)*. Le mot au début de l'arc est la tête, et le mot à la fin de l'arc est le modificateur.
Par exemple, Figure 1, *Alice* (le modificateur) est le sujet de *voit* (la tête), tel que signifié par l'arc labélisé *nsubj* dirigé entre ces 2 mots. Aussi, *Bob* (le modificateur) modifie *voit* (la tête) dans une relation d'objet direct (label *dobj*).

La figure 2 montre l'architecture du réseau de neurones du tagger et du parser.

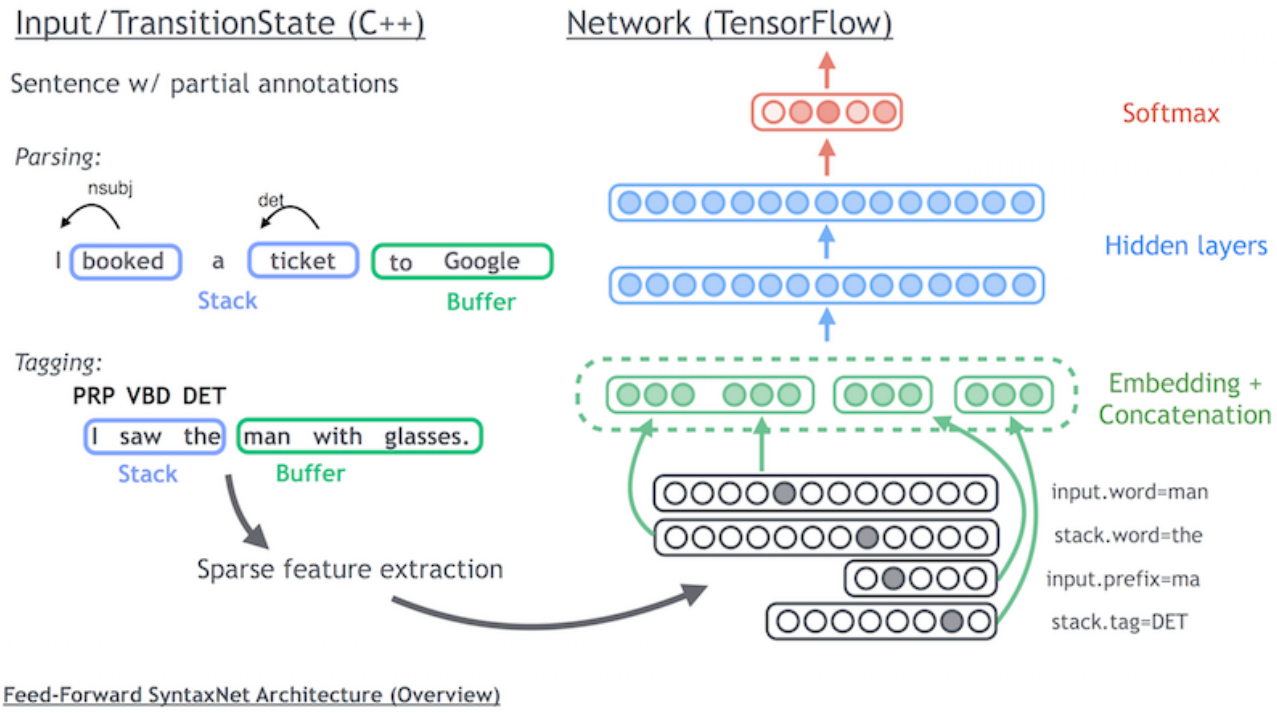

Figure 2 : Architecture du réseau de neurones de SyntaxNet

SyntaxNet est un parser de type « transition-based dependency », il implémente le système « arc-standard » définit par [Nivre, 2004]. Comme pour le tagger, le parser traite les mots de la phrase un à un de gauche à droite. En entrée, les mots non traités sont placés dans le *buffer*, et les autres mots sont placés dans la *pile* (appelée *stack*). A l'état initial, la pile contient le nœud « ROOT » et le buffer contient tous les mots de la phrase. La combinaison de la pile et du buffer est appelée la *configuration* du parser. A chaque étape, le système prend la configuration et cherche à prédire quelle est l'action à effectuer. Lorsque l'action est effectuée, on obtient une nouvelle configuration.
Les actions à prédire peuvent être une des 3 opérations suivantes :
- SHIFT : pousse un mot sur le sommet de la pile, déplaçant le premier mot du buffer vers la pile,
- LEFT_ARC : relie les 2 éléments du somment de la pile. Attache le premier (s1) au second (s2), créant un arc (s1 → s2) pointant à gauche. Sort le deuxième élément (s2) de la pile,
- RIGHT_ARC : relie les 2 éléments du somment de la pile. Attache le second (s2) au premier (s1), créant un arc (s2 → s1) pointant à droite. Sort le premier élément (s1) de la pile.
Pour chaque arc, on assigne également une étiquette (ou *label*) de relation de dépendance.
Le traitement est fini lorsqu'on arrive à la configuration suivante : le buffer ne contient plus que « ROOT » et la pile est vide.
Par exemple, pour la phrase « Alice voit Bob » de la Figure 1, le système a prédit les transitions suivantes :

Transition	Stack	Buffer	Arc-label
	[ROOT]	[Alice voit Bob]	
SHIFT	[ROOT Alice]	[voit Bob]	
SHIFT	[ROOT Alice voit]	[Bob]	
LEFT_ARC(nsubj)	[ROOT voit]	[Bob]	nsubj(voit,Alice)
SHIFT	[ROOT voit Bob]	[]	
RIGHT_ARC(dobj)	[ROOT voit]	[]	dobj(voit,Bob)
RIGHT_ARC(ROOT)	[ROOT]	[]	root(ROOT,voit)

Table 1 : Prédiction des transitions

Voici des exemples de features utilisés par le parser :
'input.token.tag '
'input(1).token.tag '
'input.token.word '
'stack.token.tag '
'stack.child(1).token.tag '
'stack.token.word '
'stack.child(1).token.word '
'stack.child(1).label '
'stack(1).child(1).label '
Les features « input » portent sur les éléments du buffer et les features « stack » sur les éléments de la pile. « child » correspond au *modifier* de l'arc, « token.word» correspond au mot du token, « token.tag » correspond au POS tag du token, et « label » fait référence au nom de l'arc.
Les features peuvent aussi porter sur les éléments dans une fenêtre, par exemple stack(1) et input(1) portent sur l'élément voisin du mot considéré respectivement dans la pile et dans le buffer.

Métrique

Les 3 métriques les plus fréquemment rencontrées permettant d'évaluer les résultats d'un parser syntaxiques sont les suivantes :
- LAS (Label Attachment Score) : pourcentage de tokens pour lesquels le système a prédit correctement les POS tags et les arcs.
- UAS (Unlabeled Attachment Score) : pourcentage de tokens pour lesquels le système a prédit correctement les arcs.
- LA (Label Accuracy) : pourcentage de tokens pour lesquels le système a prédit correctement les POS tags.

Corpus

Le corpus d'apprentissage et de test de notre modèle SyntaxNet français sont les données en français du corpus Universal Dependencies v1.3. Le lien vers le corpus est le suivant :
http://universaldependencies.org

Les données françaises d'Universal Dependencies proviennent du corpus Universal Google v1.2, qui est composé d'un mélange de phrases tirées de Google Actualités, de Blogger, de Wikipedia et de Google Local Reviews.
Les données françaises contiennent 391107 tokens, 402197 mots et 16448 phrases. Elles ont été séparées pour 88,61 % en corpus d'apprentissage, 9,64 % en corpus de développement, et 1,74 % en corpus de test.

Les listes des POS tag et des labels des dépendances utilisées dans ce corpus et donc dans le modèle SyntaxNet français peuvent être trouvées dans les liens suivants :
http://universaldependencies.org/fr/pos/all.html
http://universaldependencies.org/fr/dep/all.html

Comparaison SyntaxNet vs MaltParser

Nous avons comparé les résultats de parsing obtenus avec 2 systèmes différents : le système SyntaxNet et le système MaltParser. Les 2 systèmes utilisent la méthode « arc standard » de [Nivre 2004], et leur modèle français a été appris sur les mêmes données extraites du corpus Universal Dependencies. A la différence de SyntaxNet qui utilise des réseaux de neurones, MaltParser utilise un classifieur à régression logistique.

Le modèle français de MaltPaser a été entraîné après une étape d'optimisation des features à l'aide de l'outil MaltOptimizer.
Les données de test sont la partie « test » du corpus Universal Dependencies.
Les résultats obtenus sont présentés Table 2.

	LA	UAS	LAS
MaltParser	88,00%	80,80%	76,90%
SyntaxNet	87,70%	81,30%	76,70%

Table 2 : Comparaison des résultats entre SyntaxNet et MaltParser sur nos données de test

Nous observons que **les différences de performances entre les 2 systèmes sont négligeables, tandis que Syntaxnet présente l'avantage d'avoir une licence totalement permissive** puisque basé sur TensorFlow.

Installation de SyntaxNet

Les instructions suivantes ont été testées sur Ubuntu 16.04. Pour tout autre OS, se référer à la section « Installation » du lien https://github.com/tensorflow/models/tree/master/syntaxnet.

Lancer les commandes suivantes dans un terminal pour installer Python 2.7 :

```
sudo apt-get install python2.7
sudo apt-get install python-pip
```

Installer Bazel 0.3.2 (les versions plus récentes 0.4.x ne sont pas compatibles avec SyntaxNet) :
Aller sur la page https://github.com/bazelbuild/bazel/releases et télécharger le fichier bazel-0.3.2-installer-linux-x86_64.sh.
Lancer les commandes suivantes :

```
chmod +x bazel-0.3.2-installer-linux-x86_64.sh
./bazel-0.3.2-installer-linux-x86_64.sh --user
export PATH="$PATH:$HOME/bin"
```

Vous pouvez ajouter la ligne :
export PATH="$PATH:$HOME/bin"
dans le fichier ~/.bashrc

Lancer les commandes suivantes pour installer les outils nécessaires à SyntaxNet :

```
sudo apt-get install swig
sudo pip install --upgrade pip
sudo pip install -U protobuf==3.1.0
sudo pip install asciitree
sudo pip install numpy
sudo pip install mock
```

Pour installer SyntaxNet, lancer les commandes suivantes :

```
mkdir tensorflow
cd tensorflow
sudo apt-get install git
git clone --recursive https://github.com/tensorflow/models.git
cd models/syntaxnet/tensorflow
./configure
cd ..
bazel test syntaxnet/... util/utf8/...
```

Cette dernière commande devrait prendre plusieurs minutes.

Utilisation

Apprentissage (facultatif) :

Télécharger le corpus Universal Dependencies à l'adresse http://universaldependencies.org, section « Download » (fichier ud-treebanks-v1.4.tgz).

Lancer les commandes suivantes :

```
mkdir corpus
cd corpus
```

Copier l'archive ud-treebanks-v1.4.tgz dans le répertoire ~/tensorflow/models/syntaxnet/work/corpus et lancer les commandes :

```
tar -zxvf ud-treebanks-v1.4.tgz
cd ..
cp -r corpus/ud-treebanks-v1.4/UD_French/ .
cp -r models models_french
cp UD_English/context.pbtxt UD_French/context.pbtxt
```

Ouvrir le fichier UD_French/context.pbtxt et remplacer lignes 39, 46 et 53 « UD_English » par « UD_French » et « en-ud » par « fr-ud »

Créer le script de training français à partir du script de training anglais :

```
cp train.sh train_french.sh
```

Ouvrir le fichier train_french.sh et remplacer les 2 lignes suivantes :
l. 116 : CORPUS_DIR=${CDIR}/UD_English → CORPUS_DIR=${CDIR}/UD_French
l. 120 : MODEL_DIR=${CDIR}/models → MODEL_DIR=${CDIR}/models_french

Pour lancer l'apprentissage du modèle français, lancer la commande :

```
./train_french.sh -v -v
```

Préparer le script de test français :

```
cp test.sh test_french.sh
```

Remplacer la ligne suivante :
l. 15 : MODEL_DIR=${CDIR}/models → MODEL_DIR=${CDIR}/models_french

L'étape « **Apprentissage »** est facultative car l'archive syntaxnet.tar.gz fournie contient déjà le répertoire work/ avec le script de test français final et le modèle français final appris par le LIG.

Test :

Préalablement à cette étape, tel que décrit dans la section précédente, lancer l'apprentissage du modèle français OU copier dans le dossier ~/tensorflow/models/syntaxnet/ le répertoire work/ extrait de l'archive syntaxnet.tar.gz fournie.

Lancer la commande :

```
cd ~/tensorflow/models/syntaxnet
```

Pour parser une phrase unique, lancer par exemple :

```
echo "Alice voit Bob" | ./work/test_french.sh ./work/models_french
```

Si plusieurs phrases sont à parser, les copier dans un fichier texte appelé par exemple « input.txt » et lancer :

```
cat input.txt | ./work/test_french.sh ./work/models_french
```

Les résultats sont sortis sous forme d'arbres. Pour les sortir au format CoNLL, commenter à partir de la ligne 50 la dernière partie CONLL2TREE dans le script work/test_french.sh :

```
  --alsologtostderr \
 | \
${CONLL2TREE} \
 --task_context=${MODEL_DIR}/context.pbtxt \
 --alsologtostderr
```

à remplacer par :

```
  --alsologtostderr #\
# | \
#${CONLL2TREE} \
#  --task_context=${MODEL_DIR}/context.pbtxt \
#  --alsologtostderr
```

Le format CoNLL est décrit ici : http://universaldependencies.org/format.html.

Exemples de sorties SyntaxNet avec le modèle français Universal Dependencies

Arbres :

Input: La Cigale , ayant chanté tout l'été , se trouva fort dépourvue quand la bise fut venue .
Parse:

```
chanté VERB ROOT
 +-- Cigale PROPN nsubj
 |   +-- La DET det
 +-- , PUNCT punct
 +-- ayant AUX aux
 +-- l'été NOUN dobj
 |   +-- tout DET det
 +-- , PUNCT punct
 +-- trouva VERB acl
 |   +-- se PRON dobj
 |   +-- dépourvue ADJ xcomp
 |   |   +-- fort ADV advmod
 |   +-- venue VERB advcl
 |       +-- quand SCONJ mark
 |       +-- bise PROPN nsubjpass
 |       |   +-- la DET det
 |       +-- fut AUX auxpass
 +-- . PUNCT punct
```

Input: Pas un seul petit morceau de mouche ou de vermisseau .
Parse:

```
morceau NOUN ROOT
 +-- Pas ADV advmod
 +-- un DET det
 +-- seul ADJ amod
 +-- petit ADJ amod
 +-- mouche NOUN nmod
 |   +-- de ADP case
 |   +-- ou CONJ cc
 |   +-- vermisseau NOUN conj
 |       +-- de ADP case
```

```
 +-- . PUNCT punct
```

Input: Elle alla crier famine chez la Fourmi sa voisine , la priant de lui prêter quelque grain pour subsister jusqu'à la saison nouvelle .
Parse:

```
crier VERB ROOT
 +-- Elle PRON nsubj
 |   +-- alla PRON nsubj
 +-- famine ADJ xcomp
 +-- Fourmi PROPN nmod
 |   +-- chez ADP case
 |   +-- la DET det
 +-- voisine NOUN dobj
 |   +-- sa DET nmod:poss
 +-- , PUNCT punct
 +-- prêter VERB conj
 |   +-- priant NUM nsubj
 |   |   +-- la DET det
 |   +-- lui PRON nmod
 |   |   +-- de ADP case
 |   +-- grain NOUN dobj
 |   |   +-- quelque DET det
 |   +-- subsister VERB advcl
 |       +-- pour ADP mark
 |       +-- jusqu'à NUM nmod
 |       +-- saison NOUN dobj
 |           +-- la DET det
 |           +-- nouvelle ADJ amod
 +-- . PUNCT punct
```

Input: Je vous paierai , lui dit - elle , avant l'août , foi d'animal , intérêt et principal .
Parse:

```
dit VERB ROOT
 +-- Je PRON nsubj
 |   +-- paierai NUM nmod
 |       +-- vous PRON case
 +-- , PUNCT punct
 +-- lui PRON iobj
 +-- - PUNCT punct
 +-- elle PRON nmod
 +-- , PUNCT punct
 +-- l'août NUM nmod
 |   +-- avant ADP case
 |   +-- , PUNCT punct
 |   +-- foi NOUN conj
 |   +-- d'animal ADJ amod
 |       +-- , PUNCT punct
 |       +-- intérêt NOUN conj
 |       +-- et CONJ cc
 |       +-- principal ADJ conj
 +-- . PUNCT punct
```

Format CoNLL :

1	La	_	DET	DET	_	2	det	_	_	
2	Cigale	_	PROPN	PROPN	_	5	nsubj	_	_	
3	,	_	PUNCT	PUNCT	_	5	punct	_	_	
4	ayant	_	AUX	AUX	_	5	aux	_	_	
5	chanté	_	VERB	VERB	_	0	ROOT	_	_	
6	tout	_	DET	DET	_	7	det	_	_	
7	l'été	_	NOUN	NOUN	_	5	dobj	_	_	
8	,	_	PUNCT	PUNCT	_	5	punct	_	_	
9	se	_	PRON	PRON	_	10	dobj	_	_	
10	trouva	_	VERB	VERB	_	5	acl	_	_	
11	fort	_	ADV	ADV	_	12	advmod	_	_	
12	dépourvue	_	ADJ	ADJ	_	10	xcomp	_	_	
13	quand	_	SCONJ	SCONJ	_	17	mark	_	_	
14	la	_	DET	DET	_	15	det	_	_	
15	bise	_	PROPN	PROPN	_	17	nsubjpass	_	_	
16	fut	_	AUX	AUX	_	17	auxpass	_	_	
17	venue	_	VERB	VERB	_	10	advcl	_	_	
18	.	_	PUNCT	PUNCT	_	5	punct	_	_	

1	Pas	_	ADV	ADV	_	5	advmod	_	_	
2	un	_	DET	DET	_	5	det	_	_	
3	seul	_	ADJ	ADJ	_	5	amod	_	_	
4	petit	_	ADJ	ADJ	_	5	amod	_	_	
5	morceau	_	NOUN	NOUN	_	0	ROOT	_	_	
6	de	_	ADP	ADP	_	7	case	_	_	
7	mouche	_	NOUN	NOUN	_	5	nmod	_	_	
8	ou	_	CONJ	CONJ	_	7	cc	_	_	
9	de	_	ADP	ADP	_	10	case	_	_	
10	vermisseau	_	NOUN	NOUN	_	7	conj	_	_	
11	.	_	PUNCT	PUNCT	_	5	punct	_	_	

1	Elle	_	PRON	PRON	_	3	nsubj	_	_	
2	alla	_	PRON	PRON	_	1	nsubj	_	_	
3	crier	_	VERB	VERB	_	0	ROOT	_	_	
4	famine	_	ADJ	ADJ	_	3	xcomp	_	_	
5	chez	_	ADP	ADP	_	7	case	_	_	
6	la	_	DET	DET	_	7	det	_	_	
7	Fourmi	_	PROPN	PROPN	_	3	nmod	_	_	
8	sa	_	DET	DET	_	9	nmod:poss	_	_	
9	voisine	_	NOUN	NOUN	_	3	dobj	_	_	
10	,	_	PUNCT	PUNCT	_	3	punct	_	_	
11	la	_	DET	DET	_	12	det	_	_	
12	priant	_	NUM	NUM	_	15	nsubj	_	_	
13	de	_	ADP	ADP	_	14	case	_	_	
14	lui	_	PRON	PRON	_	15	nmod	_	_	
15	prêter	_	VERB	VERB	_	3	conj	_	_	
16	quelque	_	DET	DET	_	17	det	_	_	
17	grain	_	NOUN	NOUN	_	15	dobj	_	_	
18	pour	_	ADP	ADP	_	19	mark	_	_	
19	subsister	_	VERB	VERB	_	15	advcl	_	_	
20	jusqu'à	_	NUM	NUM	_	19	nmod	_	_	

21	la	_	DET	DET	_	22	det	_	_	
22	saison	_	NOUN	NOUN	_	19	dobj	_	_	
23	nouvelle	_	ADJ	ADJ	_	22	amod	_	_	
24	.	_	PUNCT	PUNCT	_	3	punct	_	_	

1	Je	_	PRON	PRON	_	6	nsubj	_	_	
2	vous	_	PRON	PRON	_	3	case	_	_	
3	paierai	_	NUM	NUM	_	1	nmod	_	_	
4	,	_	PUNCT	PUNCT	_	6	punct	_	_	
5	lui	_	PRON	PRON	_	6	iobj	_	_	
6	dit	_	VERB	VERB	_	0	ROOT	_	_	
7	-	_	PUNCT	PUNCT	_	6	punct	_	_	
8	elle	_	PRON	PRON	_	6	nmod	_	_	
9	,	_	PUNCT	PUNCT	_	6	punct	_	_	
10	avant	_	ADP	ADP	_	11	case	_	_	
11	l'août	_	NUM	NUM	_	6	nmod	_	_	
12	,	_	PUNCT	PUNCT	_	11	punct	_	_	
13	foi	_	NOUN	NOUN	_	11	conj	_	_	
14	d'animal	_	ADJ	ADJ	_	11	amod	_	_	
15	,	_	PUNCT	PUNCT	_	14	punct	_	_	
16	intérêt	_	NOUN	NOUN	_	14	conj	_	_	
17	et	_	CONJ	CONJ	_	14	cc	_	_	
18	principal	_	ADJ	ADJ	_	14	conj	_	_	
19	.	_	PUNCT	PUNCT	_	6	punct	_	_	

Références

[Nivre 2004] Joakim Nivre. Incrementality in deterministic dependency parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together (ACL Workshop)*, pages 50-57, 2004.