Dependency Parsing with Backtracking using Deep Reinforcement Learning

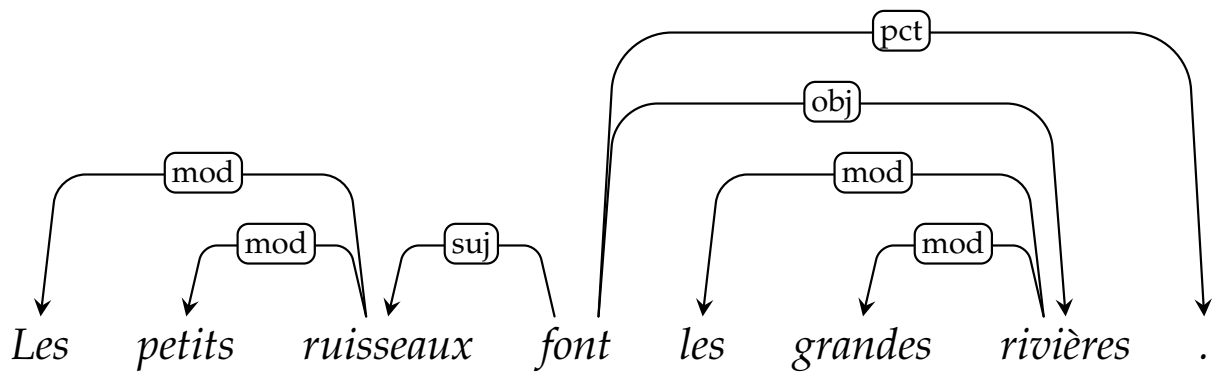
Franck Dary, Maxime Petit, Alexis Nasr

Retour Arrière

- Motivation
  - *Cette nouvelle menace est inquiétante.*
  - *Cette nouvelle menace la paix.*
- Double objectif
  - TAL : apporter une solution à un problème bien connu de l'analyse gloutonne.
  - ScCo : proposer un modèle de lecture automatique qui reproduit le comportement humain.

Analyse en transitions

stack	buffer	operation	depset
	les petits ruisseaux font les grandes rivières .	SHIFT	
les	petits ruisseaux font les grandes rivières .	SHIFT	
les petits	ruisseaux font les grandes rivières .	LEFT-*mod*	(ruisseaux, mod, petits)
les	ruisseaux font les grandes rivières .	LEFT-*mod*	(ruisseaux, mod, les)
	ruisseaux font les grandes rivières .	SHIFT	
ruisseaux	font les grandes rivières .	LEFT-*suj*	(font, suj ,ruisseaux)
	font les grandes rivières .	SHIFT	
font	les grandes rivières .	SHIFT	
font les	grandes rivières .	SHIFT	
font les grandes	rivières .	LEFT-*mod*	(rivières, mod, grandes)
font les	rivières .	LEFT-*mod*	(rivières, mod, les)
font	rivières .	RIGHT-*obj*	(font, obj, rivières)
font rivières	.	REDUCE	
font	.	RIGHT-*pct*	(font, pct, .)
font .			



Analyse en transitions

- L'analyse d'une phrase $S = w_1 \dots w_n$ est vue comme une séquence d'**actions** (appelées transitions).
- L'exécution de ces opérations construit l'arbre de dépendance corrrespondant à S.
- Une opération s'applique sur une **configuration** pour produire une nouvelle configuration.
- Une configuration $C = (\sigma, \beta, \Delta)$ où
  - β est une file qui contient initialement tous les mots de S. β_0 est le premier mot dans la file
  - σ est une pile contenant des mots de S au fur et à mesure de leur lecture. σ_0 est le mot au sommet de la pile
  - Δ est l'ensemble des dépendances construites (l'arbre de dépendances)
- Configuration initiale : $C_0 = ([], [w_1 \dots w_n], \{\})$
- Configuration d'acceptation : $(\sigma, [], \Delta)$

Classifieur

- C'est le cœur de l'analyseur.
- Il prend en entrée une configuration c et produit une distribution de probabilités sur l'ensemble de toutes les actions possibles.
- Les paramètres du classifieur sont appris sur un treebank, constitué de paires (S, A) où S est une phrase, et A est l'arbre qui lui est associé.
- Pour apprendre le classifieur, les données doivent se présenter sous la forme de paires $(c_i, a_i) \in \mathcal{C} \times \mathcal{A}$
- On transforme les arbres A_i sous la forme d'une séquence de transitions $C = c_0 \dots c_m$ avec
  1. $c_0 = C_0(A_i)$
  2. $c_m = (\sigma, [], A_i)$
- Cette transformation est réalisée à l'aide d'une fonction **oracle**.

Extension à d'autres tâches

- Le TBP est un cadre très général dans lequel on peut intégrer d'autres tâches linguistiques, il suffit de définir les actions correspondantes.
- On peut combiner les différentes actions au sein d'un même modèle.
- On peut définir un classifieur par tâche ou un classifieur pour toutes les tâches, ou pour n'importe quelle combinaison de tâches.

Analyse gloutonne

- Dans la plupart des cas, plusieurs actions peuvent d'appliquer à une configuration.
- La probabilité d'une solution est le produit des probabilités des actions qui la composent.
- La solution optimale est la solution ayant la probabilité la plus élevée :

$$\hat{s} = \arg\max_{s \in \mathcal{S}} P(s) = \arg\max_{s=a_1 \ldots a_n \in \mathcal{S}} \prod_{i=1}^{n} P(a_i)$$

- Mais l'espace de recherche $\mathcal{S}$ est de taille exponentielle par rapport à la longueur de la phrase à analyser.
- On adopte un comportement glouton :

$$\tilde{s} = \prod_{i=1}^{n} \arg\max_{a_i \in \mathcal{A}(C_{i-1})} P(a_i)$$

- La recherche de la solution est réalisée en temps linéaire, mais on n'a pas la garantie qu'il s'agit de la solution optimale.

Deux critiques

1. Le classifieur ne voit que des paires (c, a) correctes
   - Lorsque le classifieur fait une erreur, il se retrouve en terre inconnue.
   - Solutions de la littérature : oracle dynamique, apprentissage par renforcement
2. Comportement glouton : on ne revient jamais sur un choix qui a été fait
   - Si l'information obtenue plus tard dans le traitement est incompatible avec les actions réalisées, on ne peut rien y faire !
   - Solution de la littérature : beam search

Backtrack

- On ajoute au jeu de transitions une nouvelle action appelée BACK.
- BACK défait les n dernières opérations et reprend l'analyse à partir de la configuration correspondant.
- Difficulté majeure : comment prédire l'action BACK ?
  - On ne peut pas en générer à partir de l'arbre syntaxique correct grâce à l'oracle.
  - A utiliser avec parcimonie pour ne pas parcourir une partie trop importante de l'espace de recherche.

The old man the boat.

DET	ADJ			
the	old	**man**		
0	0	0	0	0

¬BACK, POS(NOUN), LEFT, LEFT, SHIFT

3	3			
DET	ADJ	NOUN		
the	old	man	**the**	
0	0	0	0	0

BACK

–	–			
DET	ADJ	–		
the	old	**man**	the	
0	0	0	0	0

BACK

–	–			
DET	–	–		
the	**old**	man	the	
0	0	1	1	0

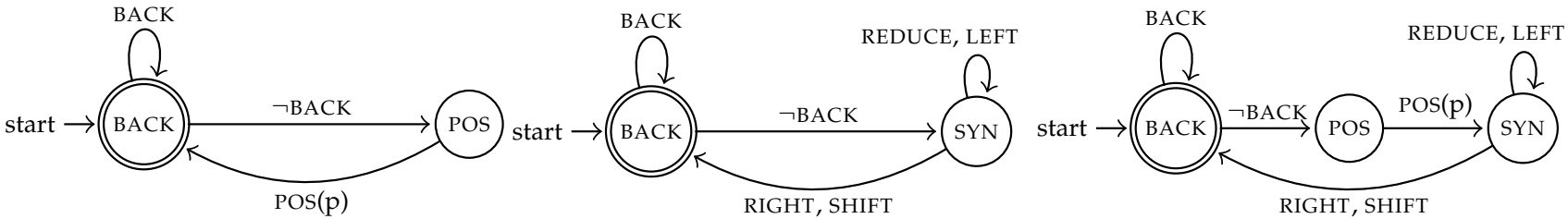
¬BACK, POS(NOUN), LEFT, SHIFT

2	–			
DET	NOUN	–		
the	old	**man**	the	
0	0	1	1	0

¬BACK, POS(VERB), LEFT, SHIFT

2	3			
DET	NOUN	VERB		
the	old	man	**the**	
0	0	1	1	0

Nouvelles architectures



- On définit de nouvelles machines pour prendre en compte l'action BACK
- Plusieurs architectures sont possibles.
- Lorsqu'une action BACK est effectuée, les dernières actions sont défaites jusqu'à atteindre une action ¬BACK.
- En d'autres termes on défait toutes les actions liées au mot précédent.

Apprentissage par renforcement

Terminologie

- Un **agent** est une entité que l'on entraîne pour prendre des décisions dans le but de réaliser un objectif.
- L'**environnement** est ce qui se trouve autour de l'agent, avec lequel il interagit.
- Un **état** définit la situation courante de l'agent.
- Les **actions** d'un agent est l'ensemble des actes qu'il peut réaliser.
- Lorsque l'agent effectue une action, l'environnement réagit en modifiant l'état et en donnant une **récompense** à l'agent.
- La **politique** de l'agent est le processus qui le mène à choisir une action.

Hypothèse de la récompense (Reward Hypothesis)

- Tout objectif peut être décrit par la maximisation des récompenses que l'agent peut accumuler.
- Apprentissage supervisé :
  - On dit à l'agent quelle action effectuer lorsqu'il est dans une situation particulière.
  - Il doit apprendre à effectuer cette action lorsqu'il se trouve dans cette situation.
- Apprentissage par renforcement :
  - On dit à l'agent ce qui se passe lorsqu'il effectue une action dans une situation particulière.
  - Il doit en déduire une politique.

Processus de décision markovien

Un MDP est un quadruplet (S, A, P_a, R_a)

- S est un ensemble d'états
- A est un ensemble d'actions
- $P_a(s, s') = P(S_{t+1} = s' | S_t = s, A_t = a)$ est la probabilité que l'action a, réalisée alors que le modèle est dans l'état s, aboutisse à l'état s'
- $R_a(s, s')$ est la récompense immédiate reçue lorsque le modèle passe de l'état s à l'état s' suite à l'action a.

Politique

- Un MDP représente l'environnement avec lequel l'agent interagit.
- Une politique $\pi(a|s)$ définit le comportement de l'agent :

$$\pi(a|s) = P(A_t = a|S_t = s)$$

- Il s'agit donc de la probabilité que l'agent choisisse l'action a s'il se trouve dans l'état s.
- Les méthodes d'apprentissage par renforcement visent à apprendre une politique à partir des échanges de l'agent avec l'environnement.

Application au transition based parsing

- Les états sont les configurations de l'analyseur.
- Les actions sont les actions de l'analyseur (SHIFT, REDUCE, RIGHT, LEFT)
- La fonction de transition $P_a(s, s')$ est déterministe : si l'on effectue l'action a dans l'état s, on aboutit à un unique état $s' = \delta(s, a)$.
- Les récompenses $R_a(s, s')$ sont calculées en fonction des dépendances créées
- Plusieurs possibilités :
  - Une récompense unique, lorsque l'analyseur atteint un état terminal (par exemple le LAS)
  - Une récompense à l'issue de chaque action

Retour Amorti

- Un épisode est une séquence d'états, d'actions et de récompense : $S_0, A_0, R_1, S_1, A_1, R_2 \ldots$
- On définit le **retour amorti** (discounted return) de la façon suivante :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- $0 \leq \gamma \leq 1$ est appelé **taux d'amortissement** (discount factor)
- γ permet de moduler l'importance des récompenses futures.
- Plus γ est élevé, plus les récompenses futures sont prises en compte.
  - Pour $\gamma = 0$, $G_t = R_t$, seule la récompense immédiate est prise en compte.
  - Pour $\gamma = 1$, les récompenses futures sont prises en compte de la même manière, quel que soit leur éloignement dans le temps.

Fonctions de valeur ($v_\pi(s), q_\pi(s,a)$)

- La valeur d'un état s étant donné une politique π, notée $v_\pi(s)$ est le retour que l'agent peut espérer obtenir s'il part de s et suit la politique π.
- On définit $v_\pi(s)$ comme l'espérance du retour amorti lorsque le modèle est dans l'état s et suit la politique π :

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

- La valeur d'une action a étant donné un état s et une politique π, notée $q_\pi(s,a)$, est le retour que l'agent peut espérer obtenir si, à partir de s, il effectue l'action a et suit la politique π.
- On définit $q_\pi(s,a)$ comme l'espérance du retour amorti lorsque le modèle choisit l'action a dans l'état s et suit la politique π :

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a]$$

Politique optimale

- Une politique π est meilleure qu'une politique π' si son retour amorti est supérieur ou égal à celui de π' pour tout état s :

$$\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v_{\pi'}(s) \; \forall s$$

- Une politique optimale, notée π_* est une politique qui est meilleure ou égale à toutes les autres politiques.

Fonctions optimales

- La fonction $v_\pi(s)$ de la politique optimale est appelée fonction de valeur optimale, notée $v_*(s)$.

$$v_*(s) = \max_\pi v_\pi(s) \; \forall s \in S$$

- On définit de la même manière la fonction d'action de valeur optimale

$$q_*(s,a) = \max_\pi q_\pi(s,a) \; \forall s \in S$$

- Les fonctions optimales spécifient les meilleurs performances que peut atteindre un MDP.

Politique optimale et fonctions optimales

- un MDP est "résolu" lorsqu'on connaît ses fonctions optimales.
- Si l'on connaît la fonction $q_*(s,a)$ alors on peut définir une politique optimale déterministe.

$$\pi_*(a|s) = \begin{cases} 1 & si\ a = \arg\max_{a'} q_*(s,a') \\ 0 & sinon \end{cases}$$

- L'objectif est donc de déterminer la fonction $q_*(s,a)$!

Q learning

- On approxime la fonction $q_*(s,a)$ à l'aide de la fonction $Q(s,a)$
- Méthode itérative
- On se trouve dans la configuration s, on effectue l'action a, le nouvel état est $\delta(s,a)$ et la récompense est R
- Comment mettre à jour $Q(s,a)$ en fonction de cette nouvelle information ?
- Nouvelle estimation de $Q(s,a)$:

$$Q'(s,a) = R + \gamma \max_{a'} Q(\delta(s,a), a')$$

- On met à jour $Q(s,a)$ avec la nouvelle estimation $Q'(s,a)$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(Q'(s,a) - Q(s,a))$$

Exploration exploitation

- Exploration : l'agent s'autorise à réaliser, à chaque étape, une action aléatoire. Il peut ainsi explorer des parties de l'espace de recherche peu fréquentées.
- Exploitation : l'agent choisit une action qui s'est révélée efficace jusque là.
- Politique ε-greedy :

$$A_t = \begin{cases} \arg\max_a Q(S_t, a) & \textit{avec une probabilité } 1 - \varepsilon \\ \textit{une action aléatoire} & \textit{avec une probabilité } \varepsilon \end{cases}$$

Deep Q-learning

- Le nombre de configurations différentes est trop important pour pouvoir les stocker dans une table $Q[s][a]$
- On approxime la fonction Q par un réseau de neurones $\hat{Q}$.
- $\hat{Q}$ prend en entrée un état s et une action a et calcule $\hat{Q}(s,a)$, une approximation de $Q(s,a)$.
- Fonction de perte :

$$L(s,a) = l_1(\hat{Q}(s,a), \hat{Q}'(s,a))$$

avec

$$\hat{Q}'(s,a) = R + \gamma \max_{a'} \hat{Q}(\delta(s,a), a')$$

Comment déterminer la fonction de récompense ?

- On s'inspire de l'oracle dynamique
- la pénalité associée à une action a est le nombre de dépendances de l'analyse correcte qui ne pourront plus être prédites si a est exécuté
- Les actions interdites ont une pénalité arbitraire (-1.5)
- La définition de la récompense de BACK est plus délicate
- Lorsqu'une action BACK est réalisée, un certain nombre d'actions $a_i \dots a_{i+k}$ sont défaites
- Chacune avait une récompense $r_i \dots r_{i+k}$.
- On définit $E = -\Sigma_{t=0}^{k} r_{i+t}$ ($E \geq 0$).
- Plus E est important, plus le nombre d'erreur l'est
- On appelle $\varphi(E)$ la fonction qui calcule la récompense d'une action BACK étant donné E.

Comment Déterminer la fonction de récompense ?

On voudrait que $\varphi(E)$ respecte les principes suivants :

1. Il ne faut pas exécuter une action BACK si aucune erreur n'a été commise : $\varphi(0) < 0$.
2. $\varphi(E)$ doit être croissante par rapport à E, plus il y a d'erreurs, plus BACK doit être encouragé
3. $\varphi(E)$ ne doit pas croître trop vite. Donner trop de récompense à une action BACK peut encourager le systèmes à faire des erreurs afin de faire ensuite des BACK et empocher le magot

Une fonction vérifiant ces principes est la suivante :

$$\varphi(E) = \begin{cases} -1 & si\ E = 0 \\ \ln(E+1) & sinon \end{cases}$$

Performances sur UD_French-GSD

	TAGGER		PARSER		TAGPARSER			
Regime	**UPOS**	p	**UAS**	p	**UPOS**	p	**UAS**	p
RLB	97.65	0.000	88.21	0.000	97.06	0.000	87.85	0.001
RL	96.84	0.000	86.60	0.037	96.73	0.090	87.12	0.211
SL	96.11	_	86.17	_	96.59	_	86.94	_

- L'apprentissage par renforcement marche mieux que l'oracle dynamique
- L'apprentissage par renforcement avec backtrack marche mieux que sans backtrack
- Les performances du Tagparser sont décevantes.

Analyse des actions BACK

	PARSER	TAGGER	TAGPARSER
#Actions	115,588	79,620	153,764
#Errs	3,597	1,323	6,249
#Backs	1,063	891	4,491
bPrec	76.86%	68.46%	73.48%
bRec	24.52%	46.49%	61.72%
C→C	18.07%	28.65%	56.89%
E→E	41.39%	26.09%	23.46%
C→E	05.15%	02.79%	02.81%
E→C	35.39%	42.47%	16.83%

- On aimerait qu'une action BACK soit déclenchée après chaque erreur et qu'elle mène à la correction de l'erreur.
- bPrec ratio du nombre d'actions BACK prédites suite à une erreur.
- bRec ratio du nombre d'erreurs après lesquelles une action BACK a été prédite.
- Mesure les capacités de détection d'erreur du système.

Quelques conclusions

	PARSER	TAGGER	TAGPARSER
#Actions	115,588	79,620	153,764
#Errs	3,597	1,323	6,249
#Backs	1,063	891	4,491
bPrec	76.86%	68.46%	73.48%
bRec	24.52%	46.49%	61.72%
C→C	18.07%	28.65%	56.89%
E→E	41.39%	26.09%	23.46%
C→E	05.15%	02.79%	02.81%
E→C	35.39%	42.47%	16.83%

- Le backtracking corrige des erreurs (E→C $\gg$ C→E).
- Le backtracking est parcimonieux. Il se déclenche rarement et, en général, après l'occurrence d'une erreur.
- Les erreurs de tagging sont plus facile à détecter et à corriger que les erreurs de parsing.

Perspectives

- Garden path sentences
  - 48 phrases
  - *Les nouvelles provoquent une grande déception.*
  - *Les nouvelles accusations sont fausses.*
  - *Cette nouvelle menace la paix.*
  - *Cette nouvelle menace est inquiétante.*
- Corrélation avec des saccades regressives ?
- Mécanisme général pour apprendre à faire du backtrack ?

Définition des transitions

Arc-gauche$_r$ construit une dépendance étiquetée r ayant pour gouverneur le premier mot de la file (β_0) et pour dépendant le mot en sommet de pile (σ_0)

$$(\sigma|w_i, w_j|\beta, \Delta) \Rightarrow (\sigma, w_j|\beta, \Delta \cup \{(w_j, r, w_i)\})$$

Arc-droit$_r$ construit une dépendance étiquetée r ayant pour gouverneur σ_0 et pour dépendant β_0

$$(\sigma|w_i, w_j|\beta, \Delta) \Rightarrow (\sigma|w_i|w_j, \beta, \Delta \cup \{(w_i, r, w_j)\})$$

Shift enlève β_0 de la file pour le mettre dans la pile

$$(\sigma, w_i|\beta, \Delta) \Rightarrow (\sigma|w_i, \beta, \Delta)$$

Reduce enlève σ_0 de la pile (dépile)

$$(\sigma|w_i, \beta, \Delta) \Rightarrow (\sigma, \beta, \Delta)$$

Calcul incrémental de la moyenne

$$\begin{aligned}
\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\
&= \frac{1}{k}(x_k + \sum_{j=1}^{k-1} x_j) \\
&= \frac{1}{k}(x_k + (k-1)\mu_{k-1}) \\
&= \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})
\end{aligned}$$

- Il est donc possible de calculer μ_k à partir de μ_{k-1}.
- On peut voir $(x_k - \mu_{k-1})$ comme une erreur mesurant la différence entre la moyenne courante et la nouvelle valeur observée.
- La moyenne est mise à jour en utilisant cette nouvelle information.