Séquence vers Séquence

RCP 217

Serge Rosmorduc
serge.rosmorduc@lecnam.net
Conservatoire National des Arts et Métiers
Cédric, équipe Vertigo

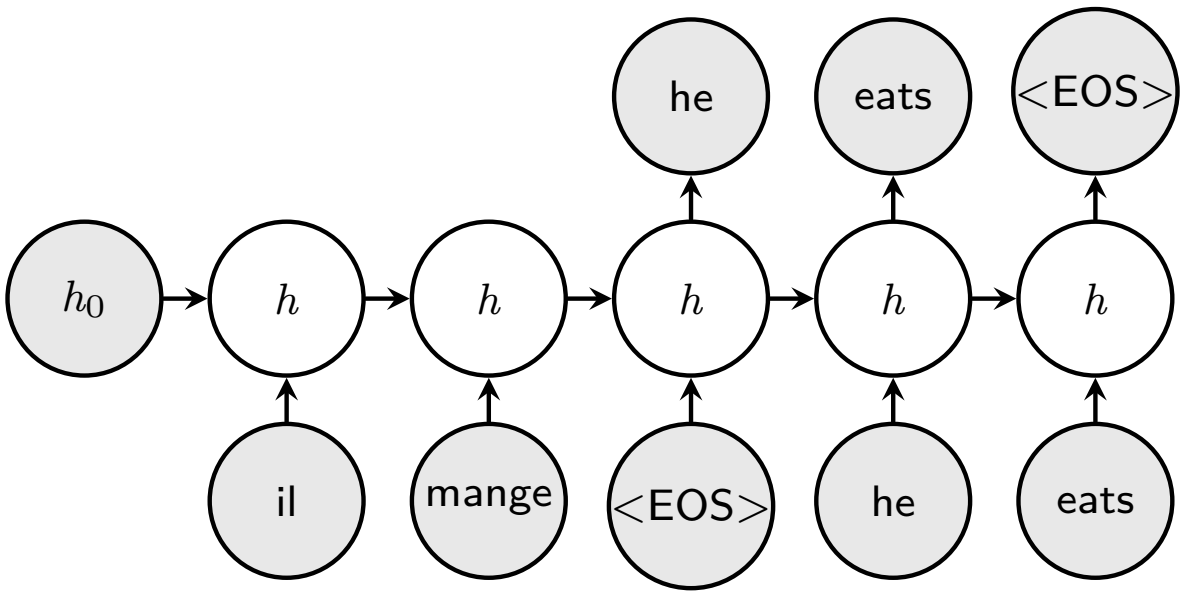
2020–2021

Modèle du langage et traduction

Sutskever, Vinyals, Le, (2014) « Sequence to Sequence Learning with Neural Networks »

Idée initiale

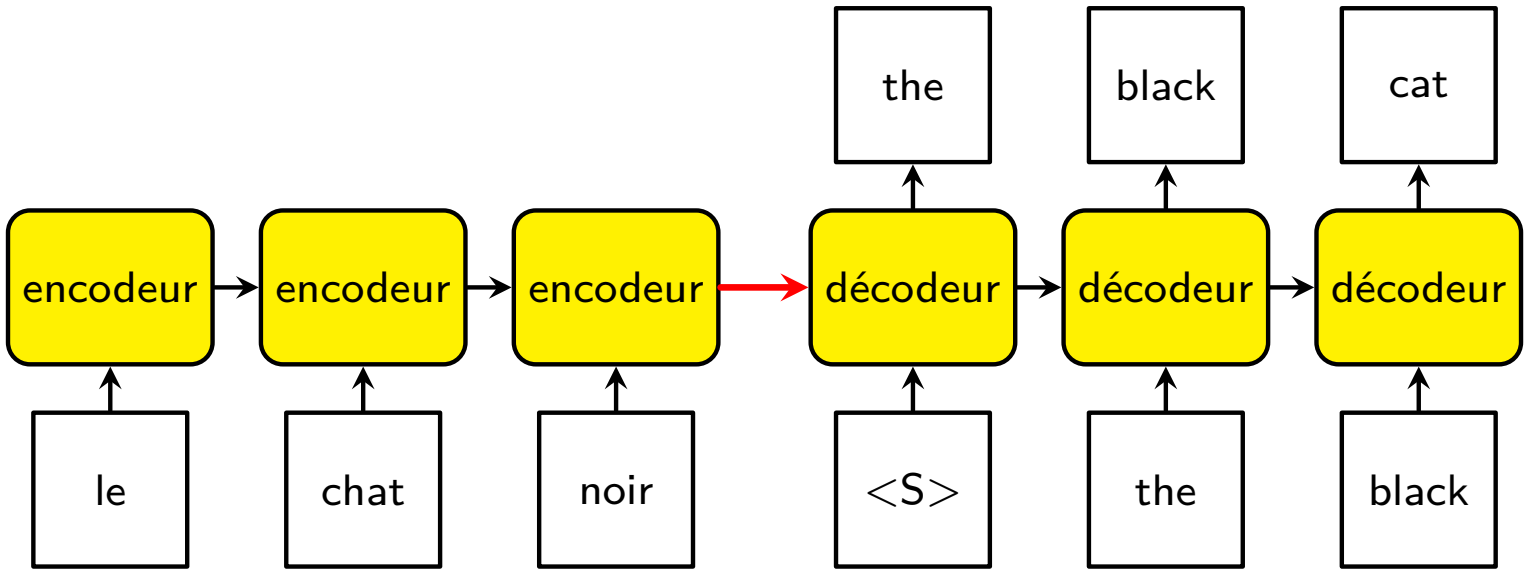
Voir la traduction comme la prédiction du prochain item dans une séquence :



En fait : architecture *encodeur/décodeur*

Architecture Encodeur/Décodeur

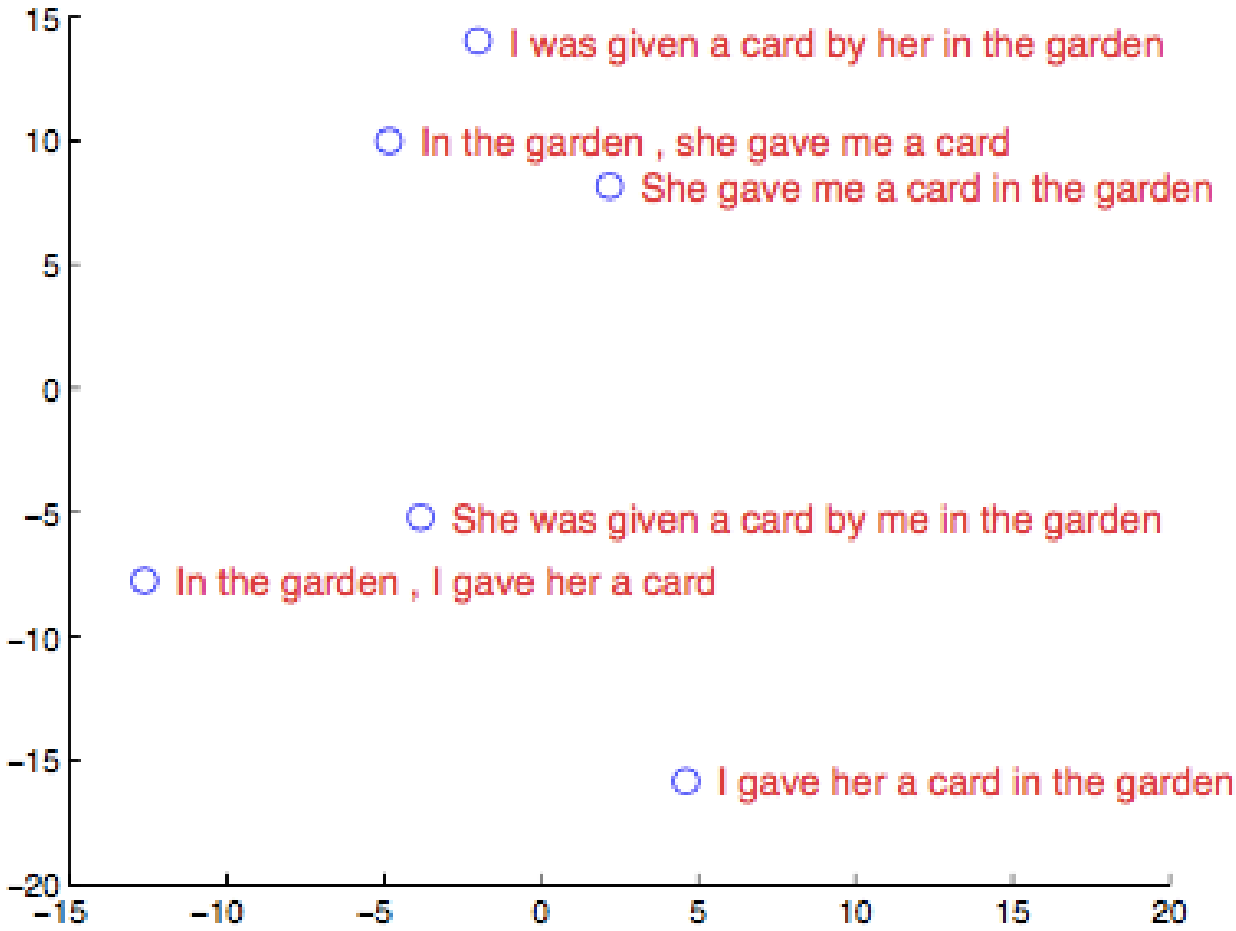
- Idée : deux réseaux récurrents ;
- le premier sert à lire l'entrée : l'encodeur ;
- on ne s'intéresse pas à sa sortie ;
- son dernier *état caché* va initialiser l'état caché $h0$ du second réseau ;
- le second réseau est le décodeur ;
- son premier état caché est initialisé avec le dernier état de l'encodeur ;
- il sert à produire la sortie.



Encodeur/Décodeur

- deux articles en 2014 :
  - Cho *et al.* « Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation » ;
  - Sutskever *et al.* « Sequence to Sequence Learning with Neural Networks » ;
- Sutskever : couches de 4 LSTM, de 1000 cellules par couche, vocabulaire d'entrée de 160 000 mots, de sortie de 80 0000 ; 380 000 000 de paramètres ;
- **renouvellement complet de la traduction automatique ;**
- on peut inverser la phrase à traduire (ça dépend des langues) et obtenir un meilleur résulat ;
- le système de Sutskever faisait presque aussi bien que les meilleurs systèmes (y compris non neuronaux) ;
- architecture **très générale,** réutilisable pour des tâches de réécritures.

Exploration des résultats de Sutskever



[Sutskever, *o.c.*, p. 6]

Fonctionnement de l'encodeur/décodeur

On a deux *sous réseaux* :

- l'encodeur prend comme entrée une phrase à encoder, et produit une sortie, qui est son *dernier état récurrent* ;
- l'encodeur peut être bi-directionnel : on connaît *tout* le texte à encoder.
- le décodeur prend en entrée :
  - un état récurrent de même dimension que celui de l'encodeur ;
  - le mot précédent de la traduction (on commence toujours par `<start>`) ;
- et produit en sortie :
  - un softmax sur le vocabulaire comme prédiction du mot suivant ;
  - un état récurrent.
- les deux seront injectés à l'étape suivante.

Décodage naïf/glouton

Quand on produit la sortie d'un système encodeur/décodeur :

- y_k dépend de $x_1, ..x_n$ *mais aussi* des choix de $y_1, ...y_k - 1$
- c'est normal : si je traduit « I like potatoes for diner », si j'ai commencé à traduire « j'aime les pommes », le mot suivant sera différent de si j'ai produit « j'aime les patates » ;

Algorithme naïf glouton :

```
x=[x0...xn]
etat = encodeur(x)
t = []
y=2 # code de <start>
faire:
  z,etat1 = decodeur(y,etat)
  y = argmax(z)
  etat = etat1
  t = t + [y]
tant que y != 3 # code de <end>
```

Décodage (suite)

- Problème : on veut la traduction **globalement** la plus probable ;
- je peux avoir un début de traduction qui a une très bonne note, suivi par une suite beaucoup moins plausible ;
- et inversement, un début jugé moins probable, mais qui sera « rattrapé » par la suite ;
- il faudrait donc explorer *complètement* l'espace des traductions...
- on ne peut même pas utiliser Viterbi : pas d'hypothèse de Markov !
- meilleure solution possible : *beam search* : on tient à jour la liste des n (n petit, souvent 5 ou 10) « meilleures » traductions connues jusqu'à présent ;

But du décodage

- $x = x_1 \cdots x_n$ phrase en entrée ;
- $y = y_1 \cdots y_m$ une traduction ;

Trouver $y' = \text{argmax}_y(P(y|x))$

	1	2	3	4	5	6
<s>	$y_1^{(0)}$	$y_2^{(0)}$	$y_3^{(0)}$	$y_4^{(0)}$	$y_5^{(0)}$	$y_6^{(0)}$
<e>	$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$	$y_4^{(1)}$	$y_5^{(1)}$	$y_6^{(1)}$
v_2	$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	$y_4^{(2)}$	$y_5^{(2)}$	$y_6^{(2)}$
v_3	$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$	$y_4^{(3)}$	$y_5^{(3)}$	$y_6^{(3)}$
v_4	$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$	$y_4^{(4)}$	$y_5^{(4)}$	$y_6^{(4)}$

possibilités de traduction

...

But du décodage

- $x = x_1 \cdots x_n$ phrase en entrée ;
- $y = y_1 \cdots y_m$ une traduction ;

Trouver $y' = \text{argmax}_y(P(y|x))$

	1	2	3	4	5	6
<s>	$y_1^{(0)}$	$y_2^{(0)}$	$y_3^{(0)}$	$y_4^{(0)}$	$y_5^{(0)}$	$y_6^{(0)}$
<e>	$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$	$y_4^{(1)}$	$y_5^{(1)}$	$y_6^{(1)}$
v_2	$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	$y_4^{(2)}$	$y_5^{(2)}$	$y_6^{(2)}$
v_3	$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$	$y_4^{(3)}$	$y_5^{(3)}$	$y_6^{(3)}$
v_4	$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$	$y_4^{(4)}$	$y_5^{(4)}$	$y_6^{(4)}$

position dans le texte engendré

But du décodage

- $x = x_1 \cdots x_n$ phrase en entrée ;
- $y = y_1 \cdots y_m$ une traduction ;

Trouver $y' = \text{argmax}_y(P(y|x))$

	1	2	3	4	5	6
<s>	$y_1^{(0)}$	$y_2^{(0)}$	$y_3^{(0)}$	$y_4^{(0)}$	$y_5^{(0)}$	$y_6^{(0)}$
<e>	$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$	$y_4^{(1)}$	$y_5^{(1)}$	$y_6^{(1)}$
v_2	$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	$y_4^{(2)}$	$y_5^{(2)}$	$y_6^{(2)}$
v_3	$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$	$y_4^{(3)}$	$y_5^{(3)}$	$y_6^{(3)}$
v_4	$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$	$y_4^{(4)}$	$y_5^{(4)}$	$y_6^{(4)}$

mot du vocabulaire

But du décodage

- $x = x_1 \cdots x_n$ phrase en entrée ;
- $y = y_1 \cdots y_m$ une traduction ;

Trouver $y' = \text{argmax}_y(P(y|x))$

	1	2	3	4	5	6
<s>	$y_1^{(0)}$	$y_2^{(0)}$	$y_3^{(0)}$	$y_4^{(0)}$	$y_5^{(0)}$	$y_6^{(0)}$
<e>	$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$	$y_4^{(1)}$	$y_5^{(1)}$	$y_6^{(1)}$
v_2	$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	$y_4^{(2)}$	$y_5^{(2)}$	$y_6^{(2)}$
v_3	$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$	$y_4^{(3)}$	$y_5^{(3)}$	$y_6^{(3)}$
v_4	$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$	$y_4^{(4)}$	$y_5^{(4)}$	$y_6^{(4)}$

hypothèse : mot v_2 en position 3

But du décodage

- $x = x_1 \cdots x_n$ phrase en entrée ;
- $y = y_1 \cdots y_m$ une traduction ;

Trouver $y' = \text{argmax}_y(P(y|x))$

	1	2	3	4	5	6
<s>	$y_1^{(0)}$	$y_2^{(0)}$	$y_3^{(0)}$	$y_4^{(0)}$	$y_5^{(0)}$	$y_6^{(0)}$
<e>	$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$	$y_4^{(1)}$	$y_5^{(1)}$	$y_6^{(1)}$
v_2	$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	$y_4^{(2)}$	$y_5^{(2)}$	$y_6^{(2)}$
v_3	$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$	$y_4^{(3)}$	$y_5^{(3)}$	$y_6^{(3)}$
v_4	$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$	$y_4^{(4)}$	$y_5^{(4)}$	$y_6^{(4)}$

But du décodage

- $x = x_1 \cdots x_n$ phrase en entrée ;
- $y = y_1 \cdots y_m$ une traduction ;

Trouver $y' = \text{argmax}_y(P(y|x))$

	1	2	3	4	5	6
<s>	$y_1^{(0)}$	$y_2^{(0)}$	$y_3^{(0)}$	$y_4^{(0)}$	$y_5^{(0)}$	$y_6^{(0)}$
<e>	$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$	$y_4^{(1)}$	$y_5^{(1)}$	$y_6^{(1)}$
v_2	$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	$y_4^{(2)}$	$y_5^{(2)}$	$y_6^{(2)}$
v_3	$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$	$y_4^{(3)}$	$y_5^{(3)}$	$y_6^{(3)}$
v_4	$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$	$y_4^{(4)}$	$y_5^{(4)}$	$y_6^{(4)}$

séquence s – v_2 – v_4 – e
e termine la séquence !

But du décodage

- $x = x_1 \cdots x_n$ phrase en entrée ;
- $y = y_1 \cdots y_m$ une traduction ;

Trouver $y' = \text{argmax}_y(P(y|x))$

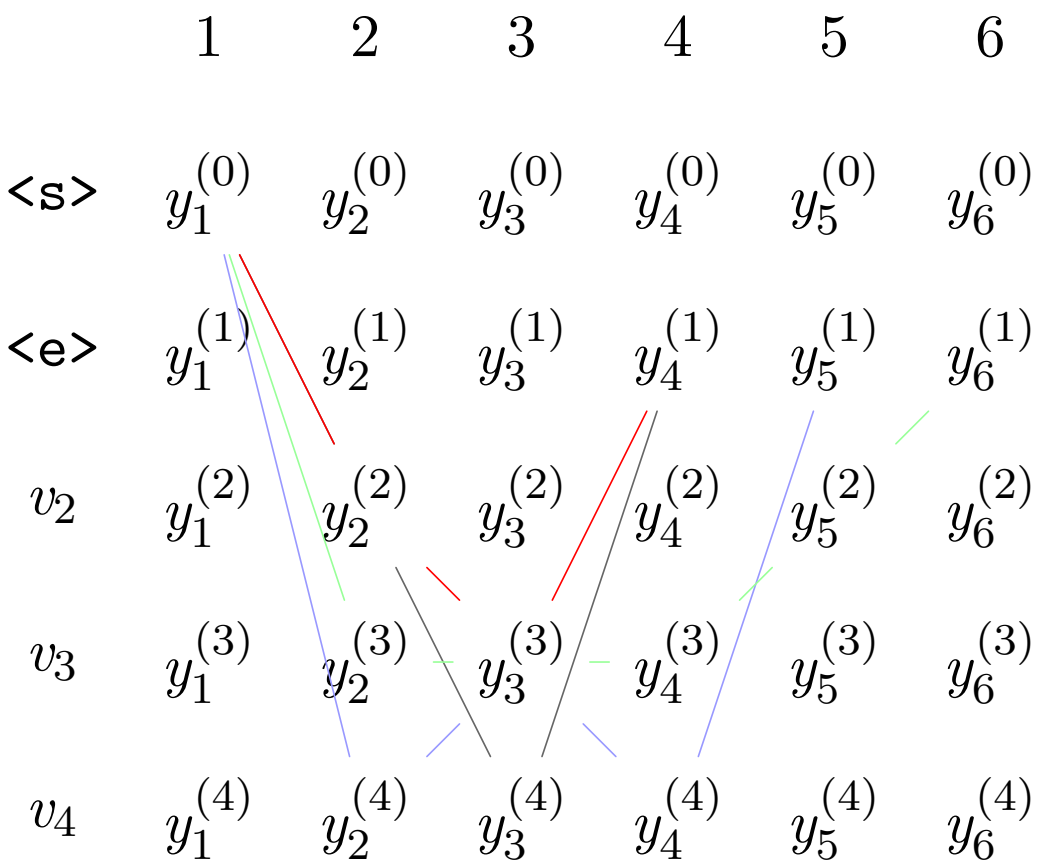
	1	2	3	4	5	6
<s>	$y_1^{(0)}$	$y_2^{(0)}$	$y_3^{(0)}$	$y_4^{(0)}$	$y_5^{(0)}$	$y_6^{(0)}$
<e>	$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$	$y_4^{(1)}$	$y_5^{(1)}$	$y_6^{(1)}$
v_2	$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	$y_4^{(2)}$	$y_5^{(2)}$	$y_6^{(2)}$
v_3	$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$	$y_4^{(3)}$	$y_5^{(3)}$	$y_6^{(3)}$
v_4	$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$	$y_4^{(4)}$	$y_5^{(4)}$	$y_6^{(4)}$

a priori, $\mathcal{O}(|v|^6)$ possibilités...

But du décodage

- $x = x_1 \cdots x_n$ phrase en entrée ;
- $y = y_1 \cdots y_m$ une traduction ;

Trouver $y' = \text{argmax}_y(P(y|x))$



but : trouver $\text{argmax}_y(P(y|x))$

Rappels

$$P(y|x) = P(y_m \cdots y_1|x) \tag{1}$$
$$= P(y_m|x, y_{m-1} \cdots y_1)P(y_{m-1} \cdots y_1|x) \tag{2}$$

Markov et Viterbi

Avec l'hypothèse de Markov, on aurait :

$$P(y_1, y_2, \dots y_m|x) = \prod_{i=1}^{m} P(y_i|x, y_{i-1})$$

- hypothèse très forte et réductrice ;
- mais fournit un algorithme (Viterbi) qui donne **la** meilleure solution modulo l'hypothèse de Markov ;
- idée : progresser niveau par niveau, en conservant le meilleur chemin qui aboutit à chaque nœud ;
- **inutilisable ici :** le réseau seq2seq nous fournit **directement** une estimation de

$$P(Y_i = v_j|x, Y_{i-1} = y_{i-1}, \cdots Y_1 = y_1, Y_0 = \text{<start>})$$

Sortie du réseau, softmax et probabilités

- le réseau a en sortie des niveau d'énergie qui donneront des probabilités *après passage dans un* `softmax` ;
- pour combiner ces probabilités, on devrait les multiplier ;
- ...ce qui numériquement fonctionne assez mal ;
- on passe au logarithme : **le produit devient une somme** ;
- **attention** ce qu'on somme, c'est le *log du softmax*, pas les énergies !
- pytorch a une fonction `log_softmax` toute faite et optimisée ;

- On veut maximiser $P(y|x)$, et donc **minimiser** $-\log P(y|x)$;
- $P(y_k, \cdots y_1|x) = P(y_k|x, y_{k-1} \cdots y_1) P(y_{k-1} \cdots y_1|x)$
- $\log P(y_k, \cdots y_1|x) = \log P(y_k|x, y_{k-1} \cdots y_1) + \log P(y_{k-1} \cdots y_1|x)$
- $\log P(y_k|x, y_{k-1} \cdots y_1)$: log softmax de la sortie du réseau à l'étape k
- $\log P(y_{k-1} \cdots y_1|x)$: valeur calculée à l'étape $k-1$

Beam search

- la recherche exhaustive est trop coûteuse ;
- solution adaptée :
  - on progresse de y_0 à la fin de la chaîne ;
  - on gère en parallèle les b meilleures hypothèses qu'on connaît ($b = 5$ par exemple) ;
  - pour chaque étape i on *étend* les b hypothèses de l'étape $i - 1$;
  - pour chaque hypothèse, cela demande d'appeler le réseau ;
  - on ne conserve que les b meilleures hypothèses.
- le cas où on choisit systématiquement le *argmax* de la sortie correspond à $b = 1$;
- le *beam search* est une heuristique ; si l'on change **un** des choix, le résultat est moins bon, mais pas de garantie si plus d'une modification.

Beam search

Le Beam (faisceau) sera une *priority queue* de taille limitée.

- elle stocke des nœuds triés (priorité, valeur) ;
- seuls les b couples de priorité minimale sont conservés.

Les opérations disponibles sont :

- ajouter(priorité, valeur) : ajoute un nœud au beam, uniquement si sa priorité le place parmi les b meilleures solutions ;
- recuperer() : renvoie les b couples conservés, triés par priorité croissante.

Pour chaque hypothèse dans le beam, on stocke :

- la traduction partielle qui lui est associée (liste d'indices de mots) ;
- **l'état des couches récurrentes du réseau après la génération de l'hypothèse**
- le beam conserve par définition le « moins log-probabilité » de l'hypothèse, comme priorité.

Condition d'arrêt

- Un nœud conservé dans le beam peut être terminal (s'il correspond à une traduction terminée par `<end>` ;
- on peut aussi décider de *limiter* la taille du résultat produit (par exemple par rapport à la taille du texte d'entrée) ;
- un beam qui ne contient plus que des nœud terminaux n'est pas extensible ;

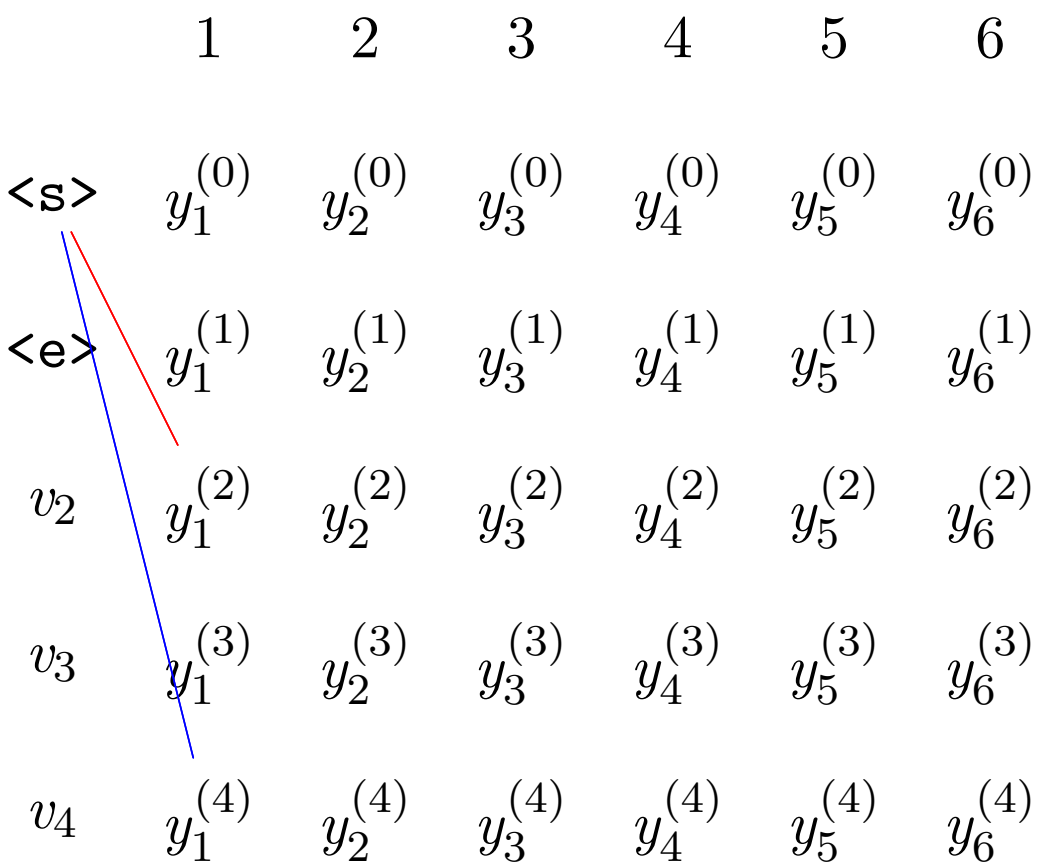
```
# pseudocode
beam = Beam() # On crée le beam de départ
lui ajouter un noeud (0, ([2], etat))
tant que le beam contient des couples extensibles
  beam_suivant = Beam() # créer le beam de l'étape suivante
  pour (logProb,noeud) dans beam.noeuds()
    si noeud est extensible  :
      t = traduction stockée dans noeud
      m = dernier mot de t
      etat = etat des couches récurrentes stocké dans noeud
      sortie, etat1 = réseau(m, etat)
      pour m1 dans les b meilleurs mots de sortie  :
         n1 = (t + [m1], etat1)
         prob = log softmax de p(m1) + logProb
         ajouter (prob, n1) à beam_suivant
    sinon
       ajouter (logProb,noeud) à beam_suivant
  beam = beam_suivant
```

Beam Search avec n = 2

	1	2	3	4	5	6
<s>	$y_1^{(0)}$	$y_2^{(0)}$	$y_3^{(0)}$	$y_4^{(0)}$	$y_5^{(0)}$	$y_6^{(0)}$
<e>	$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$	$y_4^{(1)}$	$y_5^{(1)}$	$y_6^{(1)}$
v_2	$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	$y_4^{(2)}$	$y_5^{(2)}$	$y_6^{(2)}$
v_3	$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$	$y_4^{(3)}$	$y_5^{(3)}$	$y_6^{(3)}$
v_4	$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$	$y_4^{(4)}$	$y_5^{(4)}$	$y_6^{(4)}$

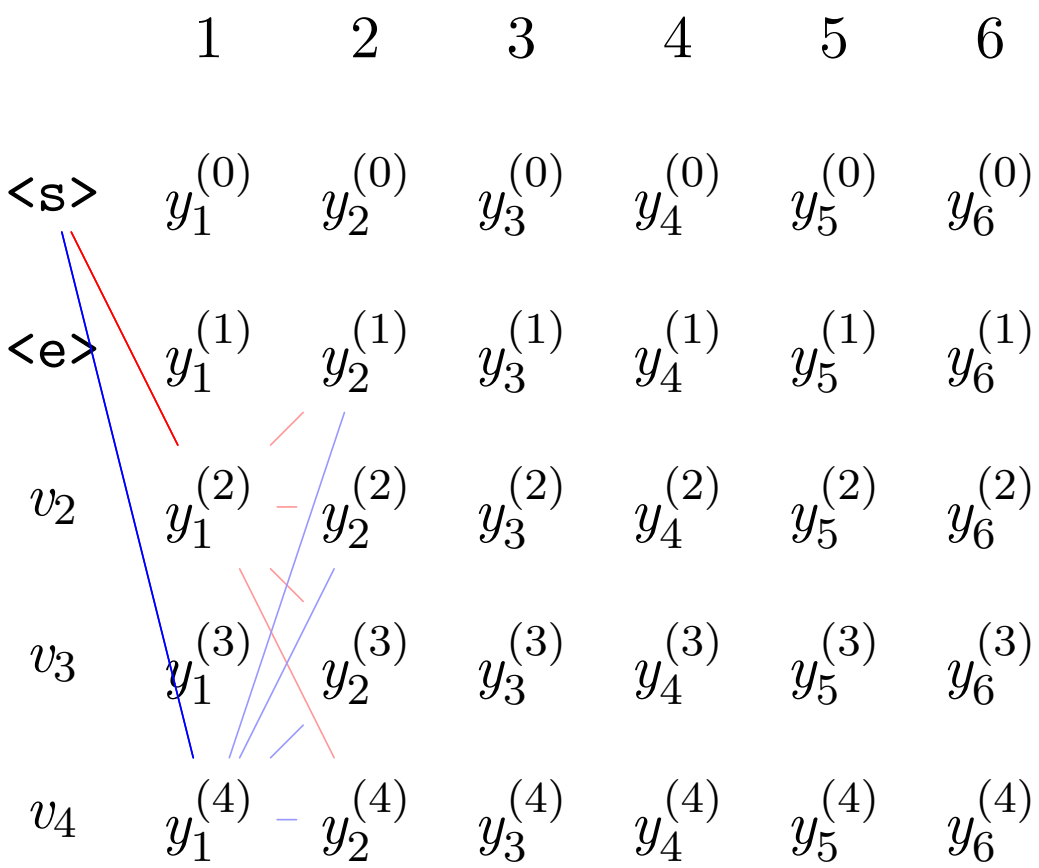
On commence à l'étape 1 : on prend les deux y les plus probables...

Beam Search avec n = 2



On commence à l'étape 1 : on prend les deux y les plus probables...

Beam Search avec n = 2



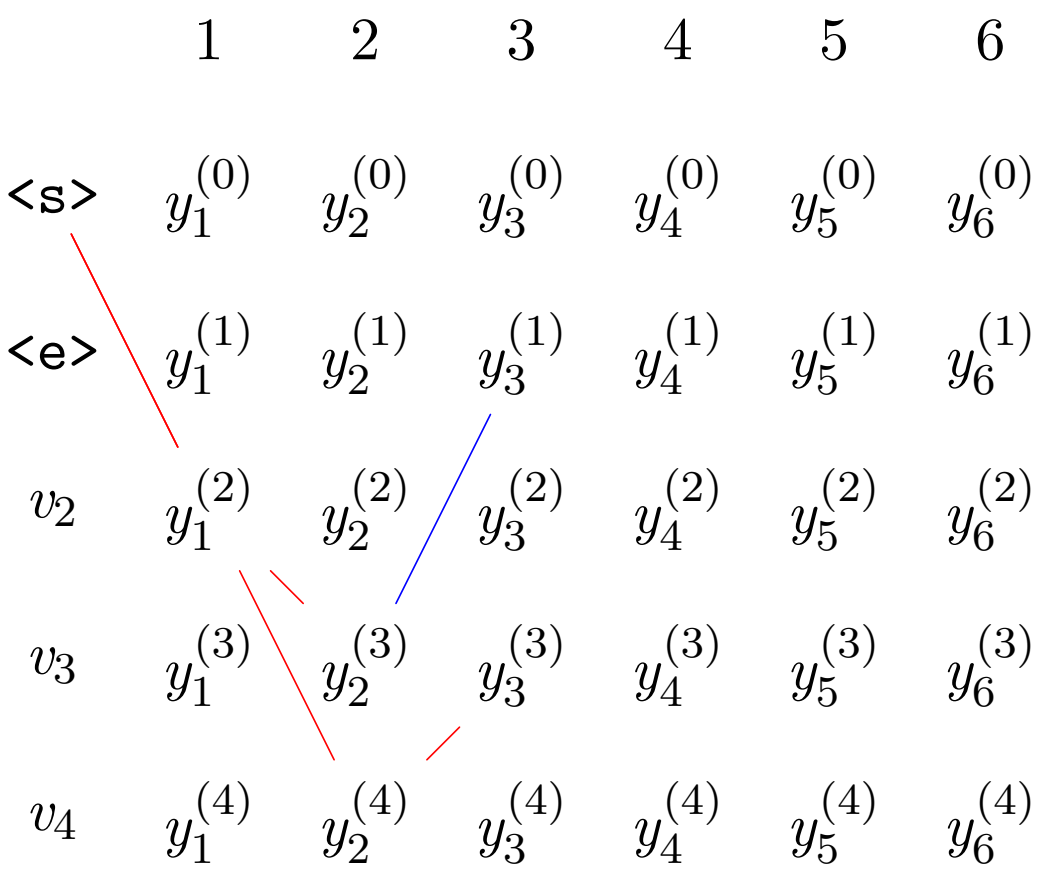
Étape 2 : 2×4 hypothèses

Beam Search avec n = 2

	1	2	3	4	5	6
<s>	$y_1^{(0)}$	$y_2^{(0)}$	$y_3^{(0)}$	$y_4^{(0)}$	$y_5^{(0)}$	$y_6^{(0)}$
<e>	$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$	$y_4^{(1)}$	$y_5^{(1)}$	$y_6^{(1)}$
v_2	$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	$y_4^{(2)}$	$y_5^{(2)}$	$y_6^{(2)}$
v_3	$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$	$y_4^{(3)}$	$y_5^{(3)}$	$y_6^{(3)}$
v_4	$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$	$y_4^{(4)}$	$y_5^{(4)}$	$y_6^{(4)}$

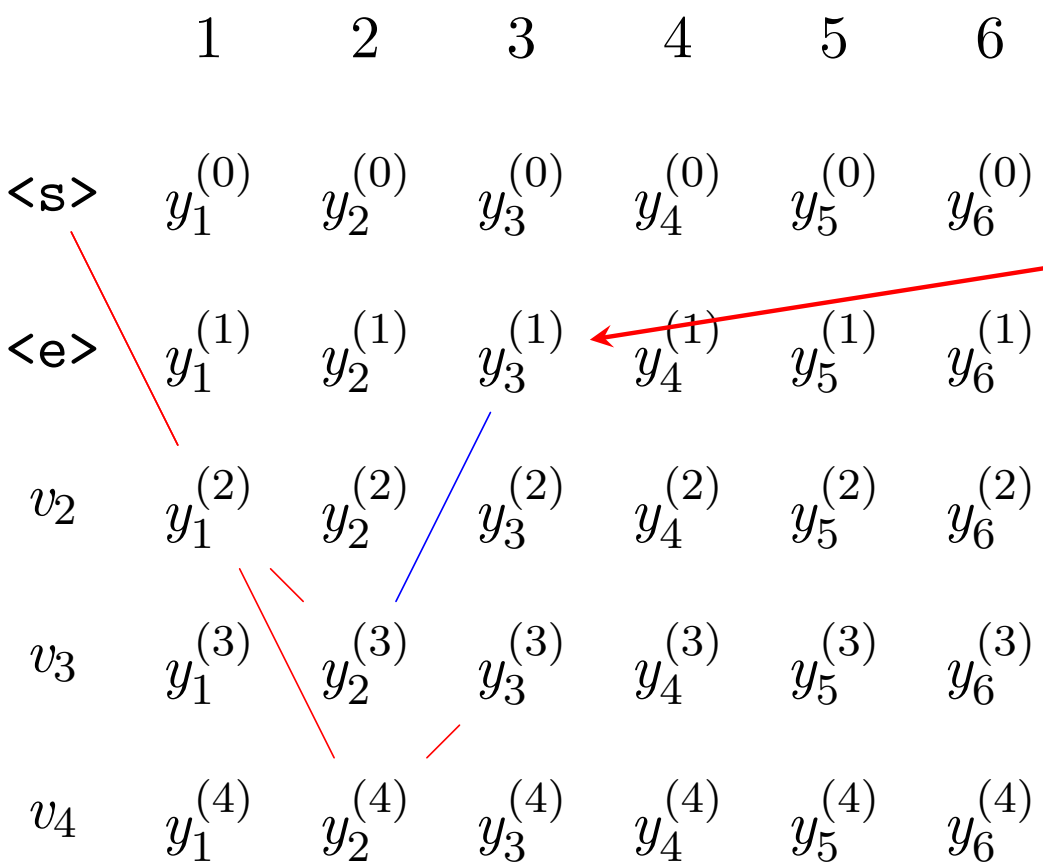
On ne garde que les deux meilleures

Beam Search avec n = 2



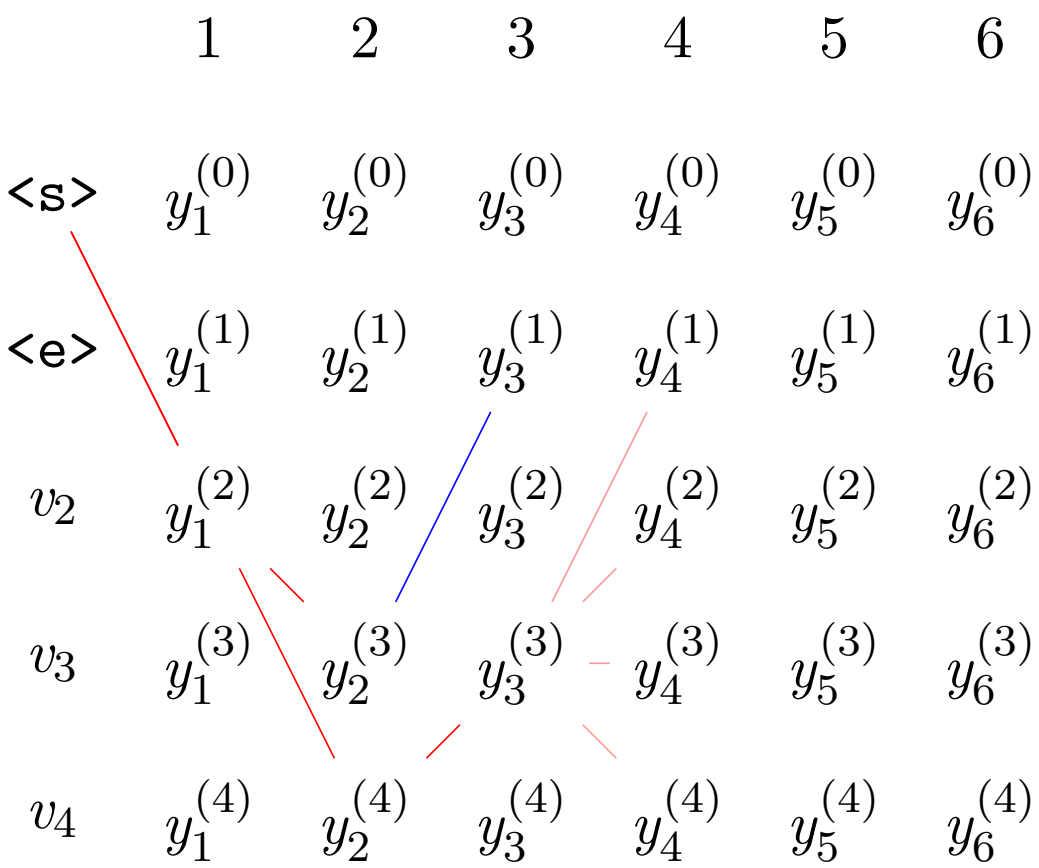
Étape 3 : même chose...

Beam Search avec n = 2



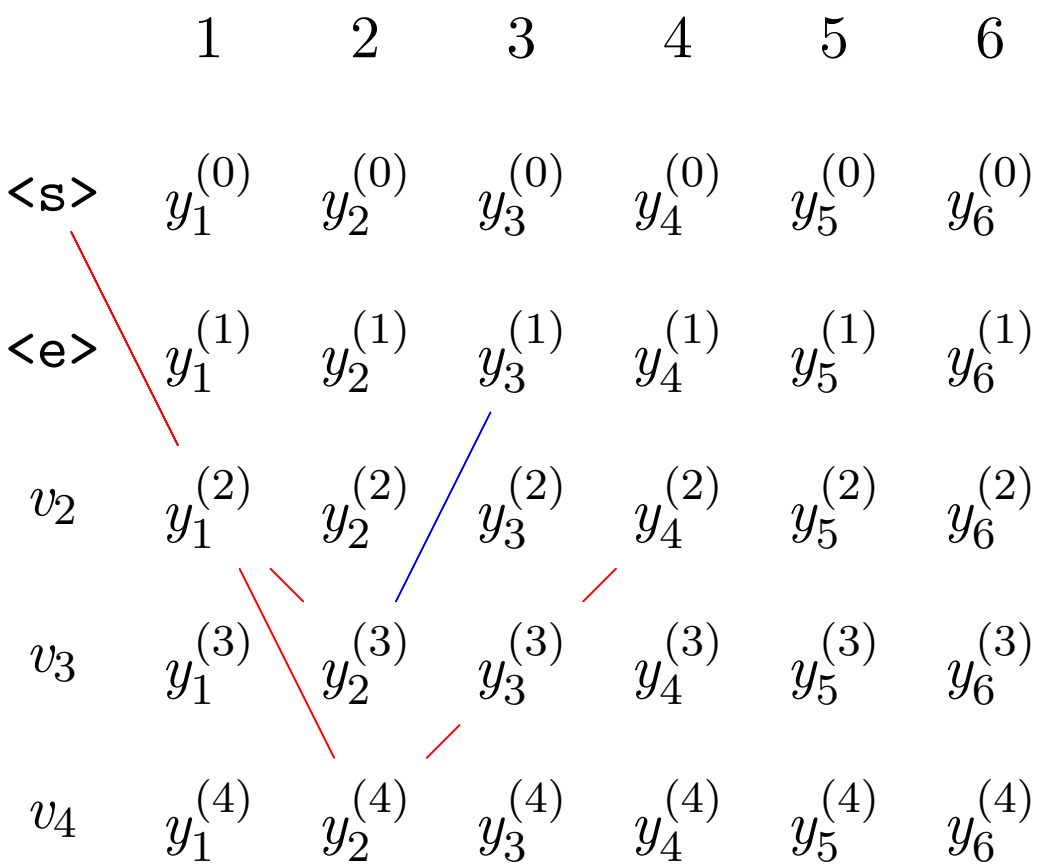
Cette hypothèse n'est plus extensible

Beam Search avec n = 2



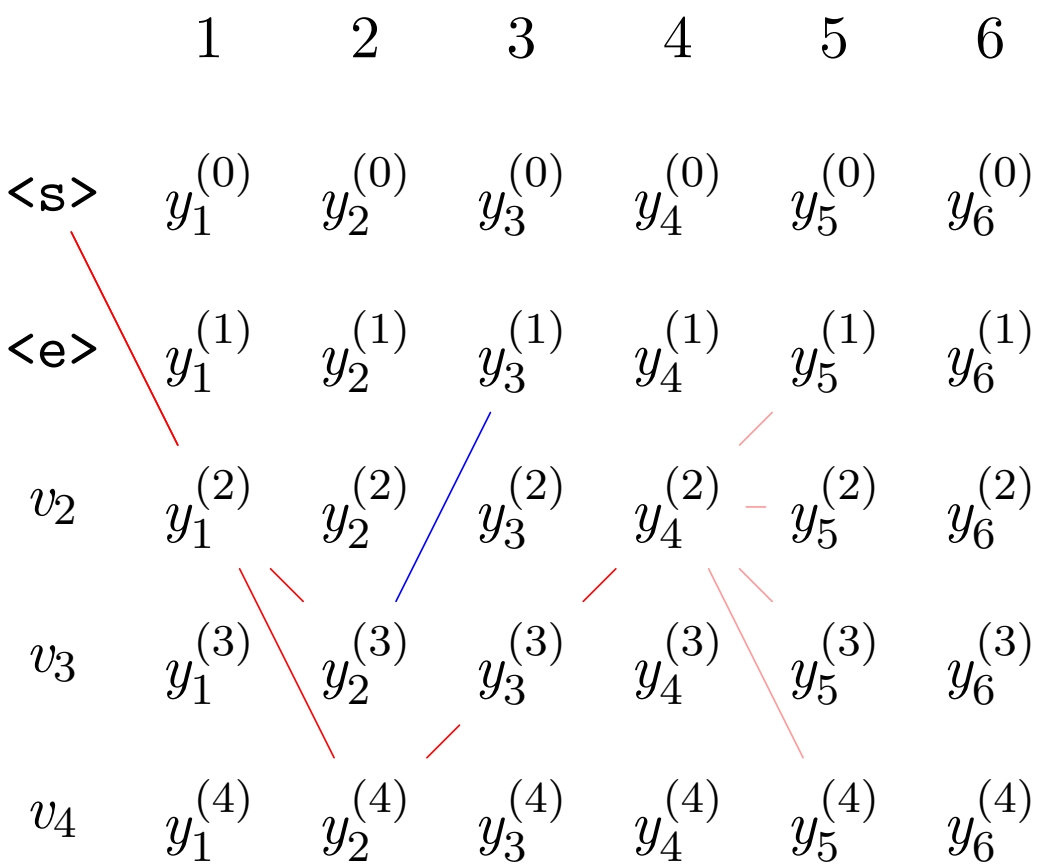
Étape 4 : on étend ce qui peut l'être

Beam Search avec n = 2



... et on conserve les deux meilleurs

Beam Search avec n = 2



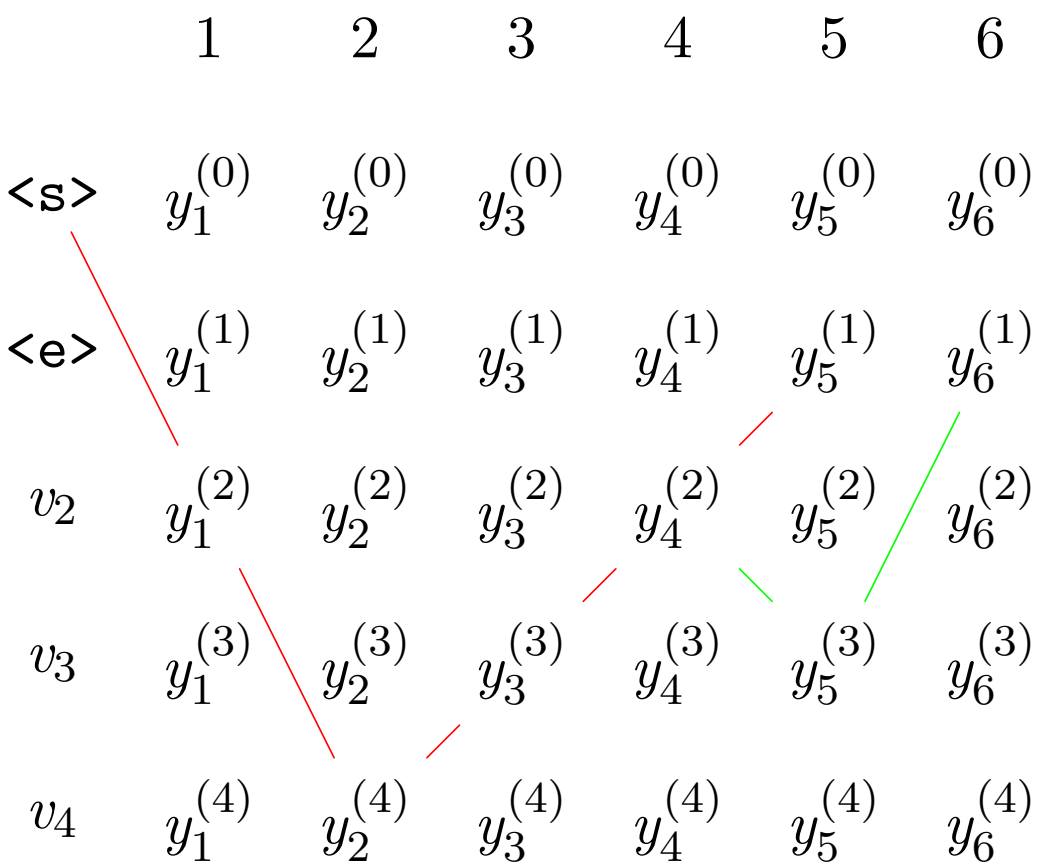
Étape 5 : etc.

Beam Search avec n = 2

	1	2	3	4	5	6
<s>	$y_1^{(0)}$	$y_2^{(0)}$	$y_3^{(0)}$	$y_4^{(0)}$	$y_5^{(0)}$	$y_6^{(0)}$
<e>	$y_1^{(1)}$	$y_2^{(1)}$	$y_3^{(1)}$	$y_4^{(1)}$	$y_5^{(1)}$	$y_6^{(1)}$
v_2	$y_1^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	$y_4^{(2)}$	$y_5^{(2)}$	$y_6^{(2)}$
v_3	$y_1^{(3)}$	$y_2^{(3)}$	$y_3^{(3)}$	$y_4^{(3)}$	$y_5^{(3)}$	$y_6^{(3)}$
v_4	$y_1^{(4)}$	$y_2^{(4)}$	$y_3^{(4)}$	$y_4^{(4)}$	$y_5^{(4)}$	$y_6^{(4)}$

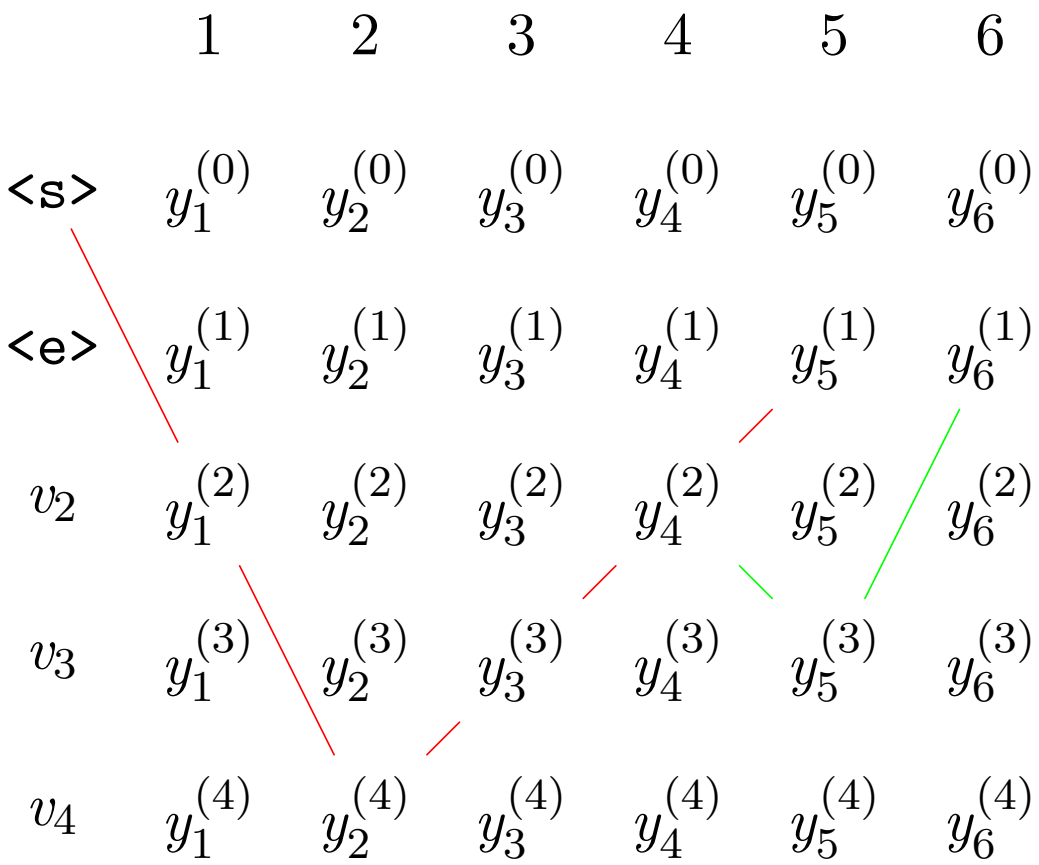
Étape 5 : etc.

Beam Search avec n = 2



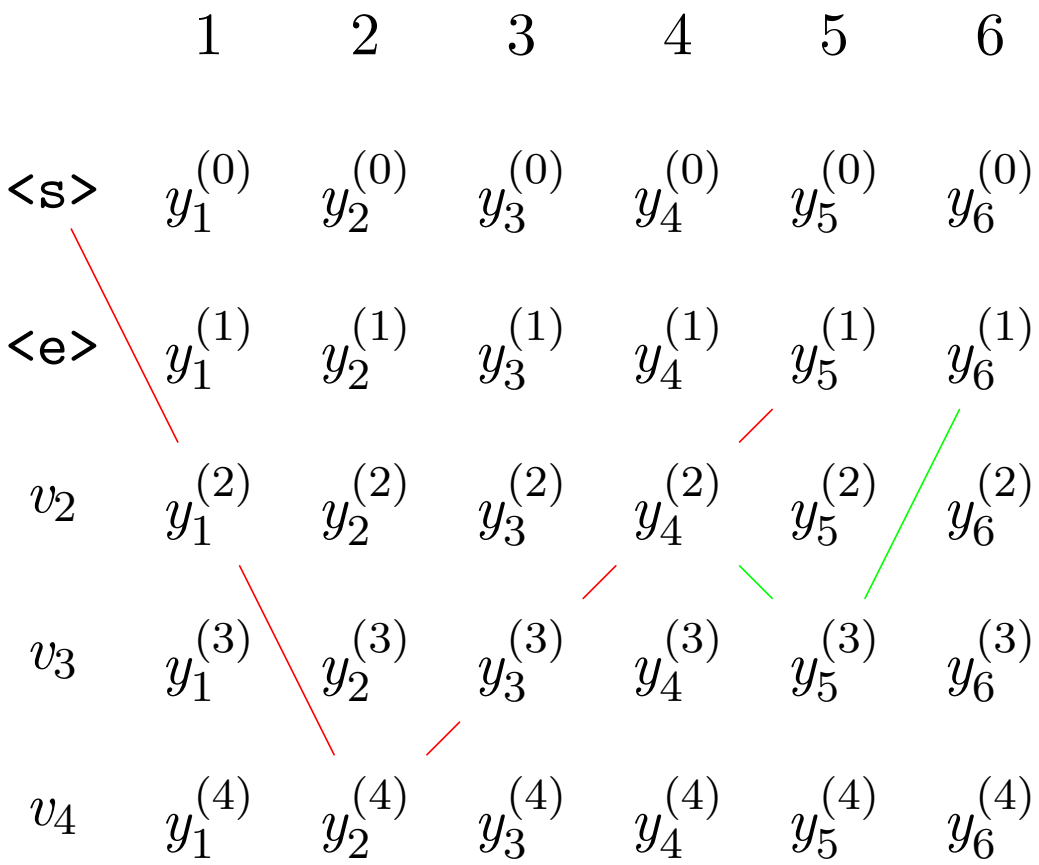
Étape 6 : etc.

Beam Search avec n = 2



Étape 6 : on s'arrête car il n'y a plus de chemin à étendre

Beam Search avec n = 2



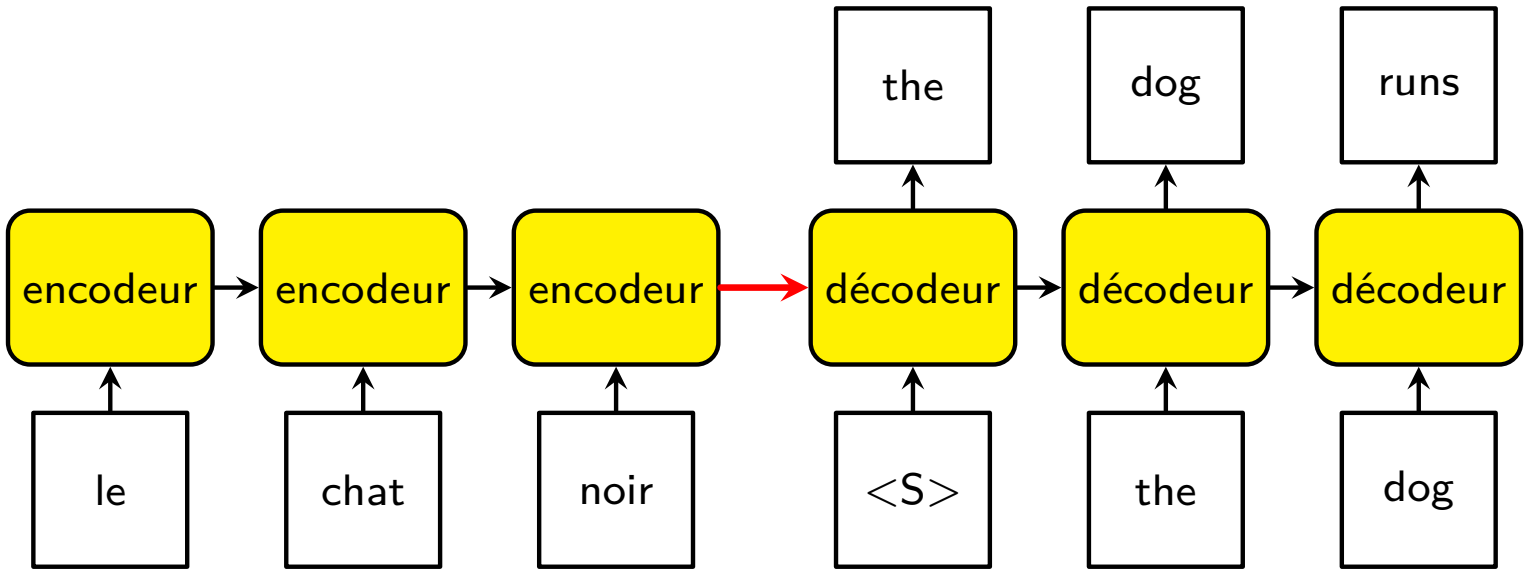
traductions :

- v2 v4 v3 v2 <FIN>
- v2 v4 v3 v2 v3<FIN>

Entraînement

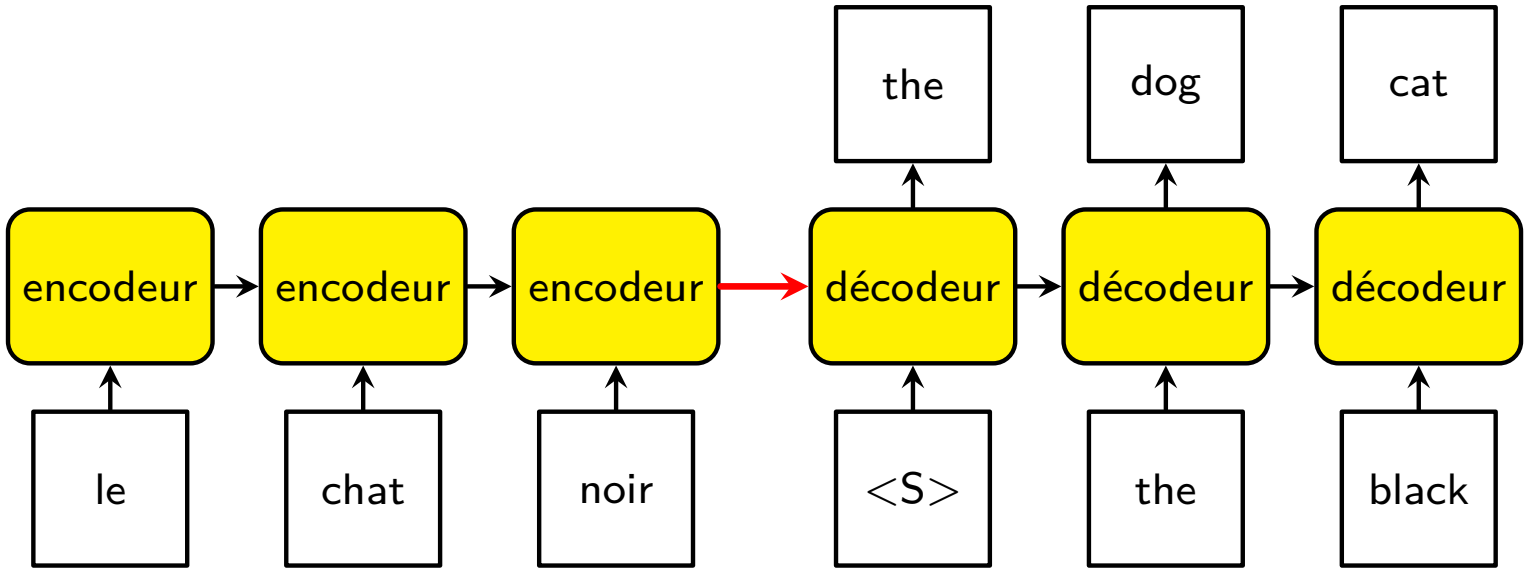
Algorithme de base :

- soit k la sortie espérée ;
- générer une traduction de longueur au plus k avec l'algorithme glouton ;
- fonction de coût : somme des *cross-entropies* pour tous les mots de la traduction *ground* ;
- générer une traduction étape par étape est long ;
- met du temps à converger.



Teacher forcing

- on prend comme *entrée* du décodeur la traduction du corpus ;
- converge plus vite que la version précédente ;
- mais ça ne correspond pas à ce que le réseau voit en production.



Mélange de teacher forcing et d'apprentissage « naïf »

- On peut commencer par quelques itérations en *teacher-forcing* puis passer à de l'apprentissage naïf ;
- On peut mélanger les deux aléatoirement
  - soit pour le traitement d'un batch en entier ;
  - soit, à chaque étape d'un décodage, choisir aléatoirement entre fournir la donnée de teacher forcing ou celle de prédite par l'étape précédente.

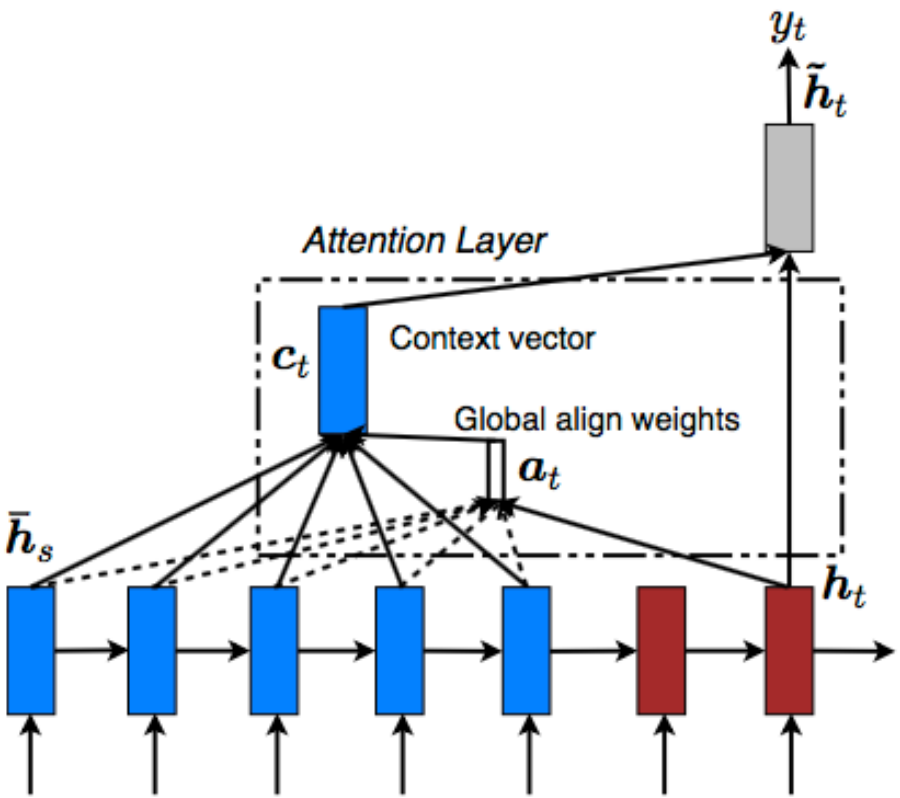
L'étape suivante : l'attention

Idées :

- l'architecture encoder/décodeur représente *l'intégralité* de la phrase d'origine par un vecteur et un seul, le dernier état du LSTM encodeur ;
- la traduction est *en grande partie* un phénomène local ;
- on aimerait bien avoir des informations plus locales.
- idée : dans le décodeur, un mécanisme d'attention nous fournit un vecteur, de somme 1, qui pondère chacun des mots de *l'entrée*.

Mécanisme d'attention

(tiré de Luong et *al.*, o.c.)



- On conserve les n vecteurs de sorties de l'encodeur ;
- Le décodeur a un premier niveau « classique » ;
- attention a_{ij} : attention portée à l'étape i du décodeur aux mot j de l'encodeur ; $\sum_{j=1}^{n} a_{ij=1}$
- combinaison en entrée de $\tilde{h}_t$ de h_t et de $c_t = \sum_{j=1}^{n} a_{tj} h_j$ par concaténation.

Modèle global d'attention, score scalaire

D'après Luong *et al.* :
On définit :

$\bar{h}_s$: état de sortie de l'encodeur à l'étape s

h_t : état de sortie, première couche du décodeur à l'étape t

$$\text{score}(h_t, \bar{h}_s) = \bar{h}_s \cdot h_t$$

enfin :

$$a_t(s) = \frac{\exp(\text{score}(h_t, \bar{h}_s))}{\sum_{s'} \exp(\text{score}(h_t, \bar{h}'_s))}$$

(On prend le softmax des scores)

En Pytorch, tensorflow, etc... ça se fait d'un bloc.

Récapitulation de l'attention globale

$$
\begin{aligned}
\text{score}(h_t, \bar{h}_s) &= \bar{h}_s \cdot h_t \\
v_{t,s} &\triangleq \text{score}(h_t, \bar{h}_s) \\
a_t &= \text{softmax}(v_t) \\
c_t &= a_t \odot s \\
\tilde{h}_t &= \tanh(\, W_c[c_t; h_t]) \\
y_t &= \text{softmax}(\tilde{h}_t)
\end{aligned}
$$

- $u \cdot v$: produit scalaire ($\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$)
- $u \odot v$: produit de Hadamard (membre à membre) ($\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$)
- En pratique, on peut se passer du $\tanh$.

Calcul de l'attention

Méthodes très variées :

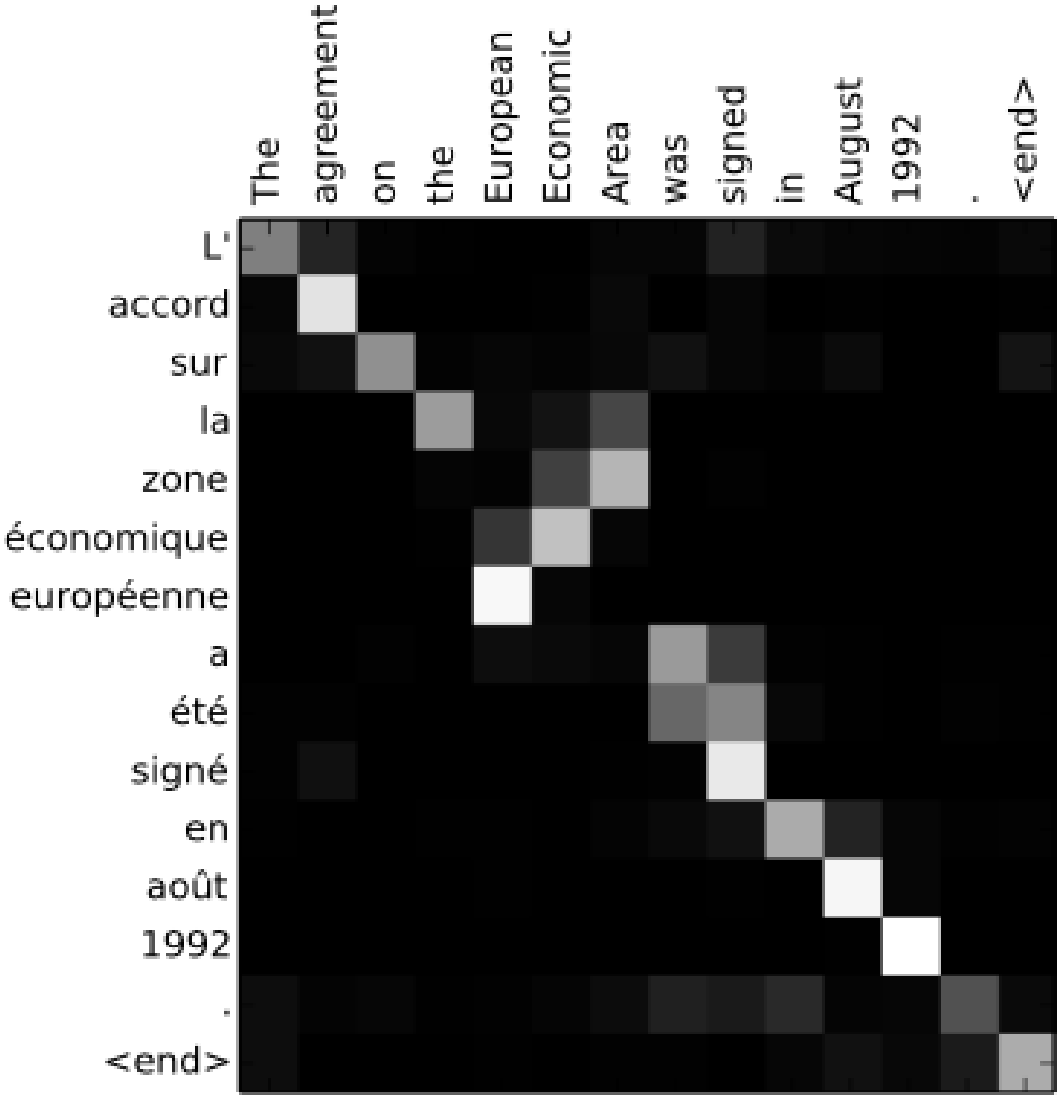
- Chez Bahdanau et al. :

$$a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

  Chaque attention α_{ij} est calculée à partir de l'état précédent s_{i-1} du décodeur et de l'état h_j de l'encodeur ;

- chez Luong *et al.* plusieurs formules d'attention sont essayées (dont celle de Bahdanau) ; les autres fonctionnent essentiellement par des formes bilinéaires : l'idée générale est que si vous avez une matrice h_e (dim $(k \times n)$) qui code les n états de l'encodeur, et une matrice h_d de dimensions $k \times m$ qui code les m états de l'encodeur, $h_e^T h_d$ est de dimensions $n \times m$, soient les dimensions nécessaires pour l'attention.
- Luong *et al.* envisagent aussi des attentions *locales* : ne portent plus sur toute la phrase d'entrée.

Visualisation de l'attention



(d'après Bahdanau *et al.*, *o.c.*)

Au delà de la traduction : la réécriture

Presque toutes les tâches peuvent se voir comme de la réécriture :

- Dialogue ;
- Résumé automatique ;
- Analyse grammaticale ;
- Calcul de dérivées ou de primitives !

Évaluations

- la précision : intéressante pour suivre l'apprentissage ;
- autre solution : la distance *Levenshtein*
- mais il peut y avoir *plusieurs* traductions possibles :
  - mesure BLEU ;
  - mesure ROUGE ;

Rappel : les n-grams

Définition

Dans un corpus découpés en mots, un *n-gram* est une suite de n mots.

Par exemple, le corpus

La rue assourdissante autour de moi hurlait

- contient les *unigrammes*, *1-gram* : { « la », « rue », « assourdissante », « autour », « de », « moi », « hurlait »}
- contient les *bigrammes, 2-gram* : { « la rue », « rue assourdissante », « assourdissante autour », « autour de », « de moi », « moi hurlait »
- en ajoutant les symboles spéciaux <DEBUT> et <FIN>, on obtient deux bi-grammes supplémentaires : « <DEBUT> la » et « hurlait <FIN> » ;
- plus n est grand, plus les *n-gram* sont signifiants, mais plus le problème de rareté des données se fait sentir.

BLEU

- Métrique comparant une traduction c à un *ensemble* R_c de traductions de références de la même phrase ;
- compare le nombre d'occurrences de *n-grams* ($n \in \{1, 2, 3, 4\}$) communs à c et aux traductions de R_c ;
- meilleur critère que la simple co-occurrence de mots isolés ;
- plus souple que la comparaison exacte ;
- peut prendre un bout d'une traduction et un bout d'une autre ;
- critère *local* et non *global* :
  - à traduire : *the cat sees the mouse*
  - ref1 : *le matou voit la souris*
  - ref2 : *le chat voit la souris*
  - à évaluer : *le matou regarde le chat*

  La solution à évaluer aura la même note que *le matou regarde la souris* !
- D'où autres métriques, plus évoluées, comme *ROUGE* (qui est une *famille* de métriques).

BLEU

- soit C un ensemble de traductions c à évaluer ;
- soit R_c un ensemble de traductions de références r pour c ;
- soit $\#_r(x)$ le nombre d'occurrences de x dans r ;
- pour un *n-gram* x :

$$\text{countclip}(x) = \min\left(\#_c(x), \max_{r \in R_c}(\#_r(x))\right)$$

(sinon, si t est « le le le le le le le », et que c est « le chat mange », on compterait 7 fois « le » !)

BLEU

Pour pénaliser la production de phrases trop courtes (une phrase courte triche : elle contient peu de *n-gram* erronés !) on calcule :

- $|C|$: longueur total des traductions produites ;
- $\mathrm{lr}(c)$: longueur de la traduction de référence dans R_c la plus proche de la longueur de c ;
- $r' = \sum_{c \in C} \mathrm{lr}(c)$: longueur théorique des traductions de référence.

BLEU

$$p_n = \frac{\sum_{t \in C} \sum_{s \in \text{n-gram}(t)} \text{countclip}(s)}{\sum_{t \in C} \sum_{s \in \text{n-gram}(t)} \#_t(s)}$$

$p_n \approx$ proportion des « bons » n-gram dans C

$$BP = \min\left(1, \exp(1 - \frac{r'}{|C|})\right)$$

pénalise le score si les traductions de |C| sont courtes

$$\text{BLEU} = \text{BP} \times \exp\left(\frac{1}{4}\sum_{i=1}^{4} \log(p_n)\right)$$

Moyenne géométrique des p_n pénalisée par BP

- surtout efficace pour comparer les évolutions d'un même système ;
- autres solutions : ROUGE, BLEURT, BERTSCORE.

Bibliographie

Les essentiels

- Sutskever, Vinyals, et Le. « Sequence to Sequence Learning with Neural Networks ». arXiv :1409.3215 [cs], 2014.
- Bahdanau, Cho, et Bengio. « Neural Machine Translation by Jointly Learning to Align and Translate ». ICLR, 2015.
- Luong M-T, Pham H., et Manning C. D.. « Effective approaches to attention-based neural machine translation » arXiv preprint arXiv :1508.04025, 2015.
- Papineni, Roukos, Ward, et Zhu. « BLEU : A Method for Automatic Evaluation of Machine Translation »,

Bibliographie

Pour aller plus loin

- Cho, van Merriënboer, Gulcehre, Bahdanau, Bougares, Schwenk, et Bengio. « Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation ». (EMNLP), 2014.
- Karakanta, Dehdari, et van Genabith. « Neural Machine Translation for Low-Resource Languages without Parallel Corpora ». Machine Translation 32, 2018.
- Lample, Ott, Conneau, Denoyer, et Ranzato. « Phrase-Based & Neural Unsupervised Machine Translation ». 2018.
- Luong, Le, Sutskever, Vinyals, et Kaiser. « Multi-Task Sequence to Sequence Learning ». ICLR, 2016.
- Vinyals, Kaiser, Koo, Petrov, Sutskever, et Hinton. « Grammar as a Foreign Language ». ANIPS, 2015.

Webographie

- Hinno, Risto. « Tuned version of seq2seq tutorial ». Medium. Consulté le 7 avril 2021.
- Trevett, Ben. bentrevett/pytorch-seq2seq. Jupyter Notebook, 2021.