BERT, BART et les Transformers

RCP 217

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

2020–2021

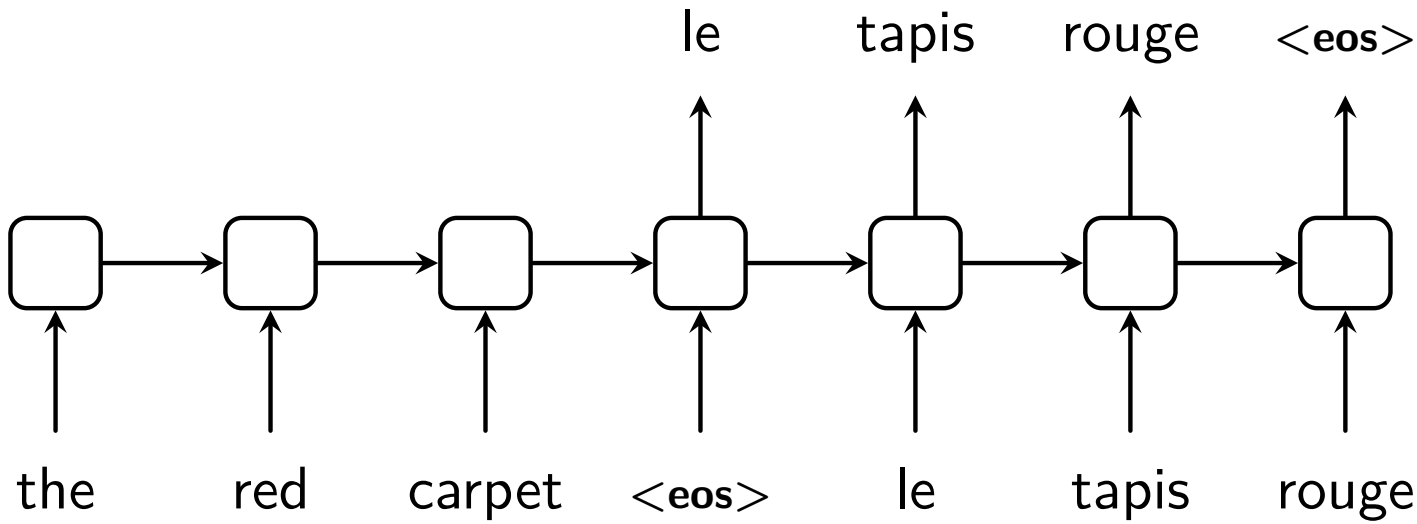
Introduction

- AlexNet (2014) : Amélioration considérable des capacités des réseaux convolutifs, saut qualitatif pour le traitement d'ImageNet ;
- possibilité de **Transfert Learning** et de **Fine Tuning** pour les images ;
- Word2Vect, Glove : embedding des *mots* ; amélioration avec **ELMO**
- **RNN** fonctionnent assez mal en **Transfert Learning**.

À partir des *Transformers,* des réseaux, **entraînés sur de très grands volumes de données**, permettent un **Transfert Learning** efficace.

Rappel : architecture Encodeur/Décodeur

Sutskever, I., Vinyals, O., et Le, Q. (2014). « Sequence to sequence learning with neural networks » . *In Advances in Neural Information Processing Systems* (NIPS 2014).



En pratique :

- un LSTM pour lire le texte en entrée : l'encodeur ;
- un LSTM pour produire la sortie : le décodeur ;
- La mémoire du décodeur est initialisée au départ avec le dernier état de l'encodeur.

Limitations

- Le dernier état de l'encodeur (un vecteur de taille fixe) représente la *totalité* du texte en entrée ;
- gère difficilement de longues phrases.

Ajout de l'attention

Bahdanau S., Cho K et Bengio Y « Neural machine translation by jointly learning to align and translate » CoRR, abs/1409.0473, 2014.
Luong M-T, Pham H., et Manning C. D.. « Effective approaches to attention-based neural machine translation » arXiv preprint arXiv :1508.04025, 2015.

- Le décodeur construit la traduction en utilisant la totalité des états intermédiaires de l'encodeur ;
- au lieu d'utiliser le dernier état h_n, on utilise une combinaison linéaire de tous les états, $\sum_{i=1}^{n} a_i h_i$
- le vecteur unitaire a est *l'attention* ;
- il est calculé à chaque étape de la production de la sortie

Le Transformer

attention is all you need

Motivation

- Les mécanismes à base de réseaux récurrents sont lents à entraîner ;
- la rétropropagation du gradient se fait sur toute la séquence
- ils se parallélisent assez mal ;
- en pratique, ils sont à la peine sur les longues phrases.

Idée

- Les mécanismes d'attention permettent de relier directement les mots **entre eux** ;
- pourquoi ne pas se passer des autres mécanismes, et n'utiliser que l'attention ?

Une vue sur l'attention

D'après la Masterclass de Łukasz Kaiser (que je vous conseille vivement)

- Attention *classique* : $h_s \cdot h_t$;
- h_t : position actuelle ; h_s : position dans texte source ;
- $\rightarrow$ distance cosinus...
- *cherche dans le passé (h_s) les éléments qui ressemblent à h_t*

Idée générale

- idée : chaque position h_s de l'entrée est associé à une valeur v_s ;
- on veut prendre le v_s du h_s qui ressemble le plus à h_t ;
- on devrait utiliser argmax, mais ça n'est pas différentiable ;
- on utilise donc $\mathrm{softmax}$!

Architecture Query/Key/Value

Query l'élément Q de départ (h_t dans notre exemple) ;

Key l'un des éléments K à comparer à Q (h_s dans l'exemple) ;

Value la valeur V qu'on veut renvoyer si K est le meilleur choix ;

L'**Attention** serait alors

$$\mathrm{A}(q, k, v) = \frac{\sum_i v_i \exp(q \cdot k_i)}{\sum_i \exp(q \cdot k_i)}$$

ou, en terme de tenseurs :

$$\mathrm{A}(Q, K, V) = \mathrm{softmax}(QK^T)\,V$$

L'attention dans les Transformers

Définition

Attention

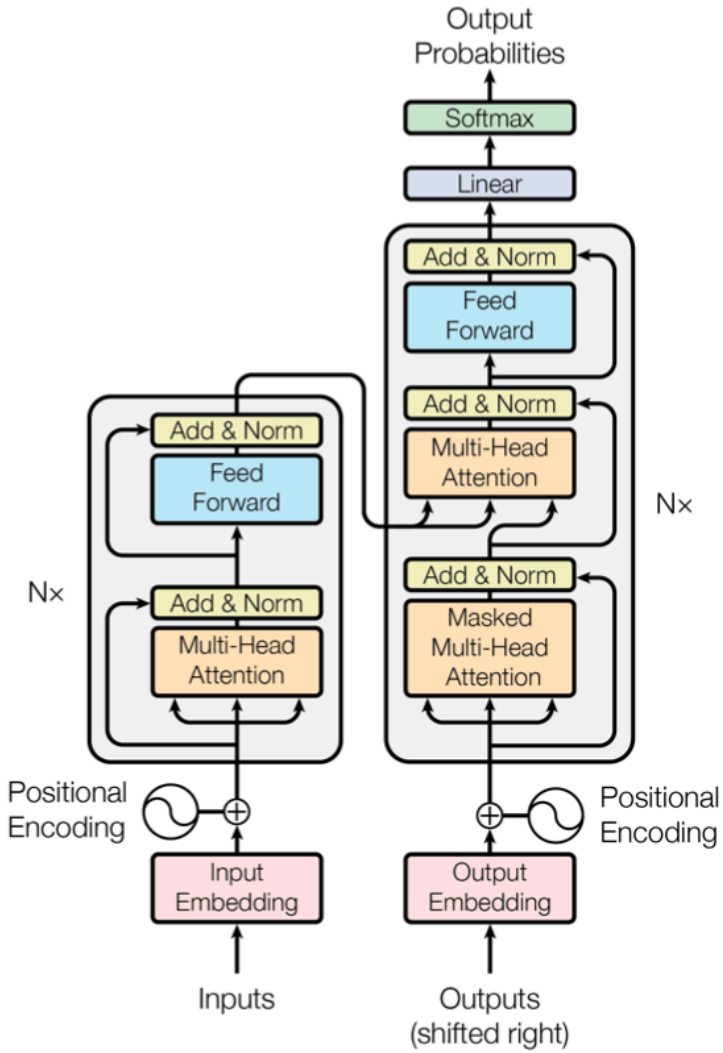
$$\text{Attention}(Q, K, V) = \left(\text{softmax}(\frac{QK^T}{\sqrt{d_k}})\right) V$$

- Q de dimensions $(n' \times d_k)$
- K de dimensions $(n \times d_k)$
- V de dimensions $(n \times d_v)$
- résultat de dimensions $n' \times d_v$

elle n'a pas de paramètres. Ceux-ci seront introduits en amont, en calculant en fait $\text{Attention}(Q_0 W^Q, K_0 W^K, V_0 W^V)$

Le Transformer

attention is all you need



Source : Vaswani et al., « Attention is All you Need ».

Entrée et couche d'embedding

- l'entrée est un vecteur d'index (entiers) de dimension variable n, chaque index correspondant à un mot du vocabulaire d'entrée ;
- l'embedding passe à une *matrice* de dimensions (n, e) ;

Différences avec Rnn

- pas de connexion récurrente !
- tous les mots de l'entrée sont traités en parallèle

Mécanisme d'attention

Utilisé **trois fois**, à partir de la même structure :

- *self-attention* dans l'encodeur : clef et query sont les représentations des mots de l'entrée **entre eux** ;
- *self-attention* dans le décodeur entre les mots déjà produits ;
- entre l'encodeur et le décodeur, *attention croisée* entre les mots de la traduction et ceux du texte initial ;
- la *self-attention* est **cruciale** :
  - pour deux mots w_i et w_j de l'entrée, l'attention a_{ij} sera le seul endroit où les informations sur les deux mots sont combinées entre elles ;
  - au contraire, dans un réseau récurrent, l'état caché h_j (j $>$ i) est produit à partir de l'état h_i, et permet donc de coder que les deux mots sont présents.

Attention multi-tête

Idée

Permettre au système d'apprendre plusieurs facteurs de décision indépendants, au lieu de tout moyenner dans *une* attention.

En pratique : on calcule plusieurs fois l'attention :

$$\begin{aligned}\mathrm{MultiHeadAttention}(Q, K, V) &= \mathrm{Concat}(\mathrm{head}_1, \cdots \mathrm{head}_h)\\ \mathrm{head}_i &= \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

On introduit une pénalisation dans la fonction de coût pour éviter que les W_i^Q (resp. K,V) soient égaux pour deux valeurs de i différentes, et ainsi représentent plusieurs phénomènes d'attention.

Intérêt du mécanisme d'attention

- complexité $O(n^2 \dot{d})$, contre $O(n \dot{d}^2)$ pour les réseaux récurrents ou convolutifs ;
- très bien parallélisables (pour l'apprentissage - en exploitation, c'est différent) ;
- pour tout couple de mots de l'entrée, quelle que soit leur distances, la profondeur des calculs qui les relie est *fixe* : le système gère plus facilement les dépendances à distance ;
- la visualisation des attentions donne (un peu) d'information sur ce que fait le modèle.

Self-Attention dans le décodeur

Soient deux mots w_i et w_j dans le **décodeur**, avec $i \leq j$

- w_j « prête attention à w_i », qui est **déjà** produit ;
- mais w_i **ne doit pas** prêter attention à w_j, qui n'est pas déjà produit quand w_i est produit ;
- on utilise un **masque** tel que a_{ij} soit à $-\infty$ pour $i < j$, ce qui donne 0 après $\mathrm{softmax}$.

Codage des positions

- Supposons qu'un même mot se retrouve plusieurs fois dans l'entrée ;
- avec le mécanisme d'attention, les lignes qui correspondent à ce mot produiront deux lignes identiques dans la sortie ;
- le mécanisme lui-même est indifférent aux positions.
- il faut donc les coder explicitement dans les données elles-mêmes.

Codage des positions

On ajoute à chaque vecteur représentant un mot un vecteur (de même dimension) codant ses coordonnées.
Soit d la dimension du vecteur PE en question, et p la position du mot ($1 \leq p \leq n$).
On prend :

$$\mathrm{PE}[p, 2i] = \sin(\frac{p}{1000^{2i/d}})$$
$$\mathrm{PE}[p, 2i+1] = \cos(\frac{p}{1000^{2i/d}})$$

Intérêt :

- fonctionne quelle que soit la taille du texte ;
- $\exists M \in L(\mathbb{R}^d, \mathbb{R}^d) \mid \mathrm{PE}(pos + k) = M\mathrm{PE}(pos)$

Chemins et complexité

Type de de Couche	Complexité	Longueur max de chemin
Self-Attention	$\mathcal{O}(n^2 \cdot d)$	$\mathcal{O}(1)$
RNN	$\mathcal{O}(n \cdot d^2)$	$\mathcal{O}(n)$

- Liaison directe systématique entre deux mots : pas de vanishing gradient ;
- complexité en $\mathcal{O}(n^2 \cdot d)$ vs. $\mathcal{O}(n \cdot d^2)$, mais généralement $n \ll d$.

BERT

Réseau formé de la partie **encodeur** d'un transformer

Pré-entraîné sur deux tâches simultanément :

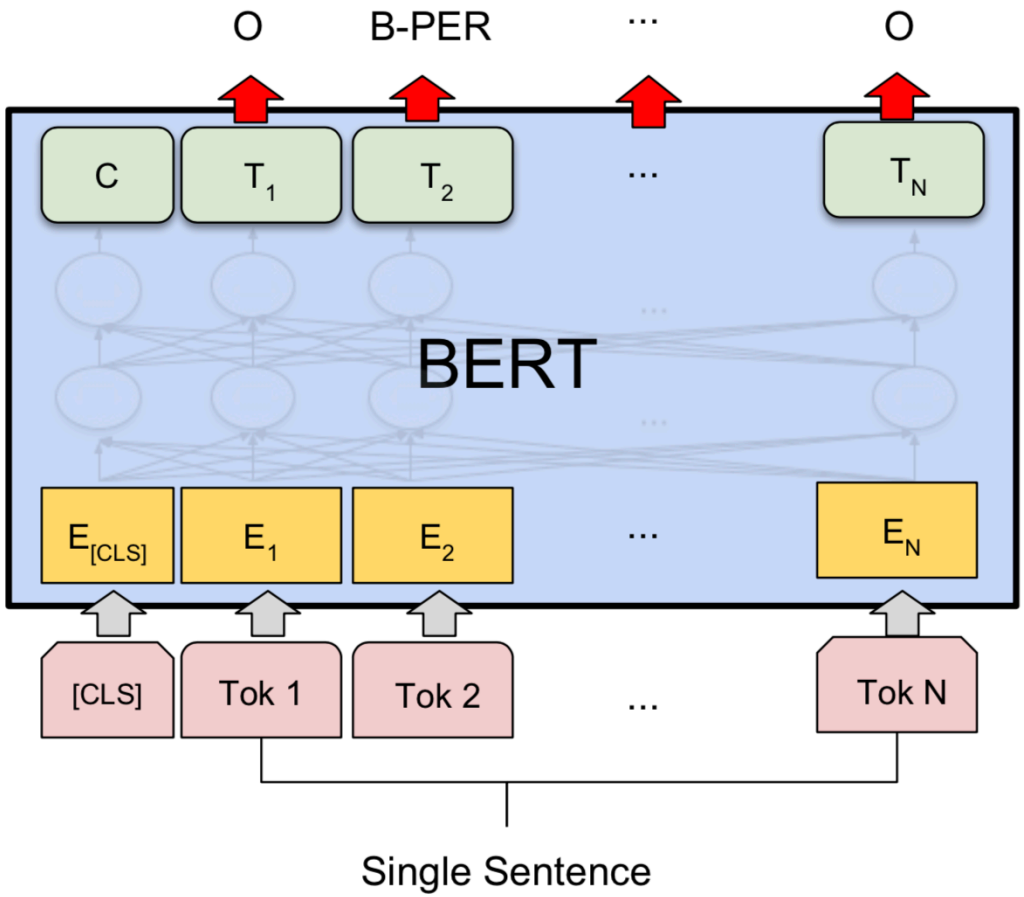
- prédiction de mots *masqués* (15 %) : on remplace aléatoirement des mots par <**mask**>, et en sortie le réseau doit les prédire ;
- prédire si deux phrases peuvent se suivre :
  - on fournit au système, soit des séquences de deux phrases consécutives du corpus ;
  - soit de deux phrases indépendantes tirées du corpus.

Utilisation de BERT

Plusieurs réseaux déjà entraînés disponibles ; utilisables :

- directement sur l'apprentissage déjà réalisé ;
- après *fine-tuning* ;
- des versions **prêtes à l'emploi sont disponibles pour certaines tâches** ;
- BERT = encodeur : pas d'utilisation très concluantes en traduction automatique.

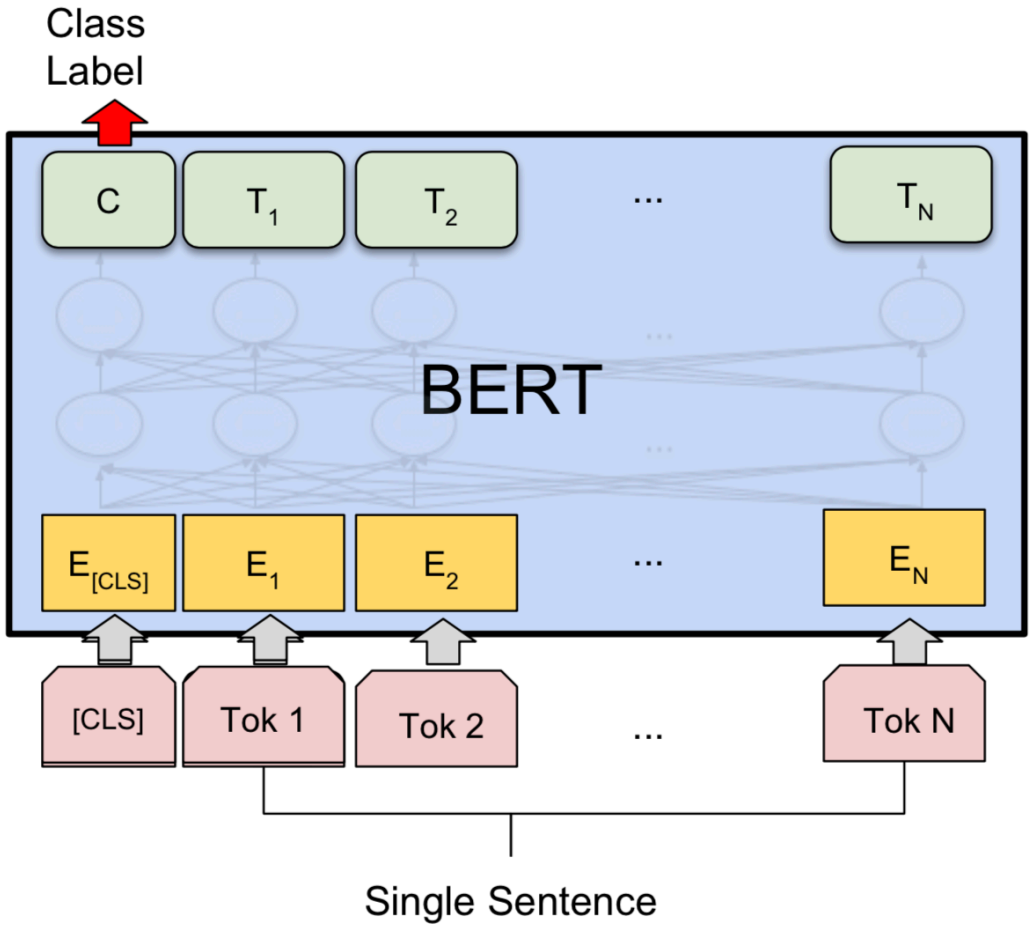
BERT pour la Named Entity Recognition



(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

Source : BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding

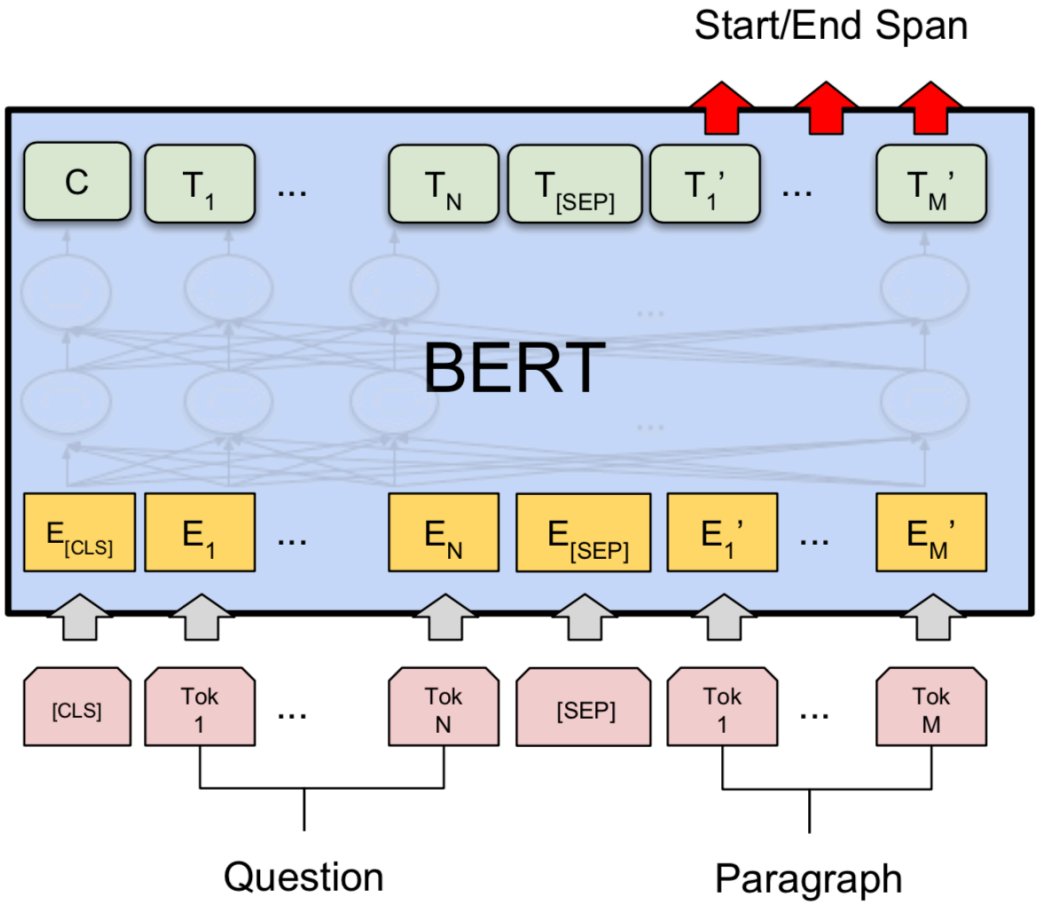
BERT pour la classification



(b) Single Sentence Classification Tasks:
SST-2, CoLA

Source : BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding

BERT pour la réponse aux questions



(c) Question Answering Tasks: SQuAD v1.1

Source : BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding

Et les autres...

ELMO embedding *contextuel* des mots

Transformer-XL transformer adapté pour traiter de très longs textes ; introduit des *positions relatives* ;

XLNet Basé sur un *modèle du langage* étendu à toutes les permutations ; comme **BERT,** c'est un encodeur ;

BART couple encodeur/décodeur ;

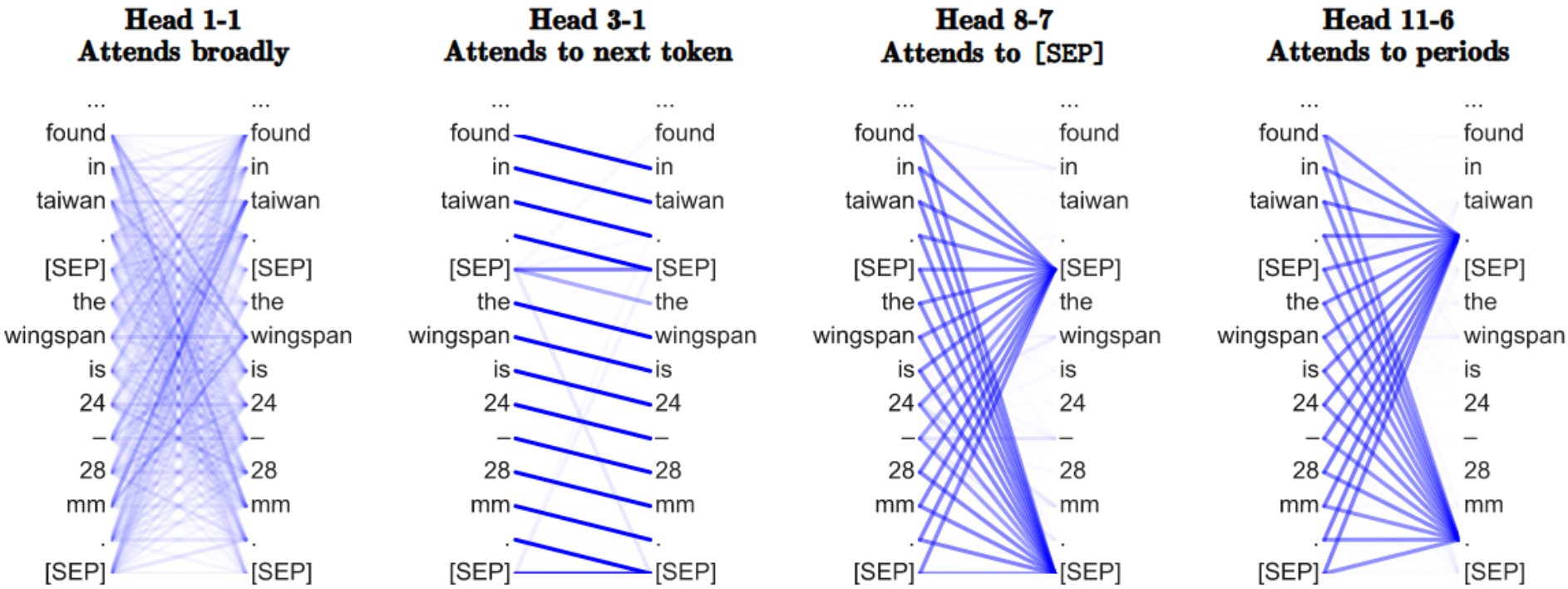
T5 réseaux *seq2seq* entraîné sur des tâches très multiples ;

CamemBERT version française de *BERT.*

et plein d'autres...

Bert et attention

Certaines têtes modélisent des relations simples :



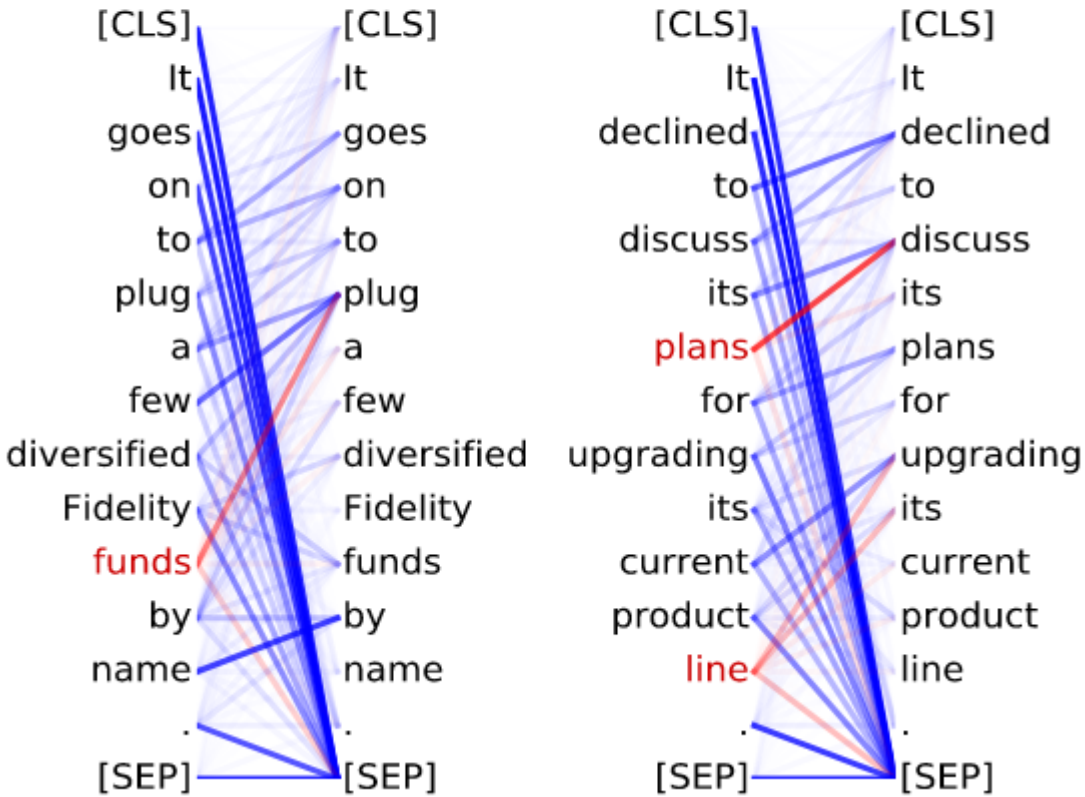
Source : (Clark et al. 2019)

Bert et attention

D'autres têtes modélisent des relations syntaxiques :

Head 8-10

- **Direct objects** attend to their verbs
- 86.8% accuracy at the `dobj` relation



Source : (Clark et al. 2019)

HuggingFace/transformers

Une des multiples bibliothèques fournissant des implémentations de tous les réseaux modernes, avec versions pré-entraînées.

- interface tensorflow ou pytorch ;
- facile à mettre en œuvre tel quel ;
- fine-tunable.

HuggingFace

- AutoClasses : idée simple : le chemin du modèle indique la classe du modèle à charger ;

  ```
  model = AutoModel.from_pretrained('bert-base-cased')
  ```

  Crée automatiquement un `BertModel` ;
- configuration avec `AutoConfig` :

  ```
  from transformers import AutoConfig

  # Change some config attributes when loading a pretrained
  config.
  config = AutoConfig.from_pretrained('bert-base-uncased',
                                      output_attentions=True)
  ```

- `AutoTokenizer`

  ```
  from transformers import AutoTokenizer
  tokenizer = AutoTokenizer.from_pretrained(
    'dbmdz/bert-base-german-cased')
  ```

Pipeline

- Modèle utilisables tels quels
- tâches : *feature-extraction*, *text-classification*, *token-classification*, *question-answering*, *table-question-answering*, *fill-mask*, *summarization*, *translation*, *text2text-generation*, *text-generation*, *zero-shot-classification*, *conversational*, *translation_XX_to_YY* ;

```
import transformers
nlp = transformers.pipeline("sentiment-analysis")
print(nlp("this is good !"))
```

['label' : 'POSITIVE', 'score' : 0.9998394250869751]

Le *Zero-shot learning*

Idée

Classification en **comparant** les documents à un ensemble d'étiquettes librement créées, sans *fine-tuning* sur celles-ci.

Exemple

```
nlpZero= pipeline("zero-shot-classification")
etiquettes = ["social event", "computing", "science",
              "fun", "sport", "theater", "art"]
texte = "I compiled my Java program, but it failed."
print(nlpZero(texte, etiquettes))
```

Question answering

Idée

Extrait d'un texte la réponse à une question

Exemple

```
nlpQa = pipeline("question-answering")
question = "where was I before London ?"
texte = "I went to Nice, Paris and then London."
print(nlpQa( question, texte))
```

N'invente pas de texte. Cite une partie du document passé en second argument.

Classification « classique »

- on prend un modèle adapté à la langue (ex. **Bert** pour l'anglais)
- on fait du *fine-tuning*
- voir TP.

Choix du modèle

Le système ne trouve pas toujours un modèle, il faut éventuellement le préciser :

```
from transformers import pipeline

nlp = pipeline("translation_fra_to_eng",
               model="Helsinki-NLP/opus-mt-fr-en")

print(nlp("Bien le bonjour à votre dame !"))
```

Catalogue pratique de modèles à l'URL
https://huggingface.co/models

Bibliographie

- Clark K. *et al.* « What Does BERT Look At ? An Analysis of BERT's Attention ». arXiv :1906.04341 [cs]
- Dai Z. *et al.* « Transformer-XL : Attentive Language Models Beyond a Fixed-Length Context ». arXiv :1901.02860
- Devlin J. *et al.* « BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding ». arXiv :1810.04805 [cs]
- Peters M. E. *et al.* « Deep contextualized word representations ». arXiv :1802.05365 [cs]
- Raffel C. *et al.* « Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer ». arXiv :1910.10683 [cs, stat]
- Vaswani A. *et al.* « Attention is All you Need ». In : Guyon I *et al.* (éd.). Advances in Neural Information Processing Systems 30
- Yang Z. *et al.* « XLNet : Generalized Autoregressive Pretraining for Language Understanding ». arXiv :1906.08237 [cs]

Webographie

- Alammar J. « The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning) ». (consulté le 18 avril 2021)
- Mihaila G. « https ://gmihaila.medium.com/fine-tune-transformers-in-pytorch-using-transformers-57b40450635 ». In : Medium, 2020. (consulté le 11 avril 2021)
- Łukasz Kaiser, Pi School. https ://www.youtube.com/watch ?v=rBCqOTEfxvg, 2017. (consulté le 15 avril 2021)
- Exploring Transfer Learning with T5 : the Text-To-Text Transfer Transformer. Google AI Blog. (consulté le 7 avril 2021a)
- « Hugging Face — The AI community building the future. ». (consulté le 18 avril 2021b)