

Le traitement du langage naturel par transformers illustré par un exemple pour la classification de texte

Cerisara Nathan, MPI

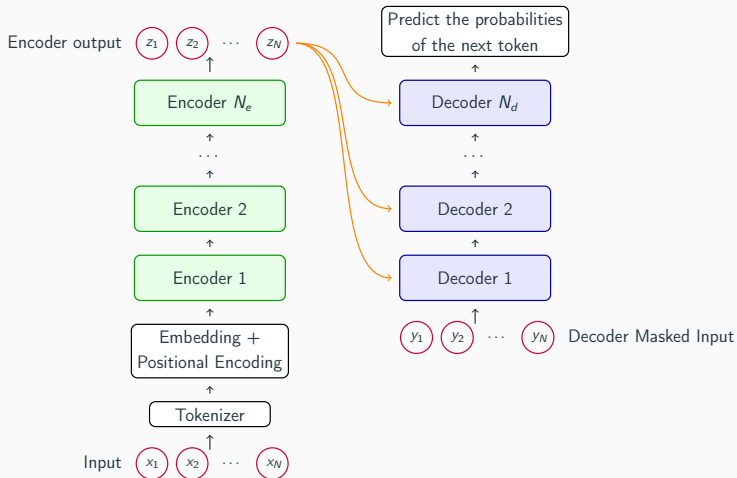
31 mai 2023

Sommaire

- Architecture Transformer
 - Vectorisation du texte
 - La partie Encodeur de l'architecture
 - Les matrices d'Attention
 - Le réseau Feed Forward
- Application personnelle
 - Objectif / Rapport à la ville
 - Le modèle BERT
 - La structure du réseau de neurone utilisée
 - Les données et l'apprentissage
 - Les résultats
- Annexes

L'architecture Transformer

Schéma de l'architecture dans le cas de la génération :



Vectorisation du texte : Tokenisation du texte

Tokenizer (bert-base-uncased)

Ex1 :

SENTENCE : "Neural Networks are so cool!"

TOKENS :

[101, 15756, 6125, 2024, 2061, 4658, 999, 102, 0, ..., 0]

[CLS] "neural" "networks" "are" "so" "cool" "!" [SEP]

Ex2 :

SENTENCE : "Bonjour le monde!"

TOKENS :

[101, 14753, 23099, 2099, 3393, 23117, 999, 102, 0, ..., 0]

[CLS] "bon" "##jou" "##r" "le" "monde" "!" [SEP]

Vectorisation du texte : Embeddings & Positional Encoding

Positional Encoding

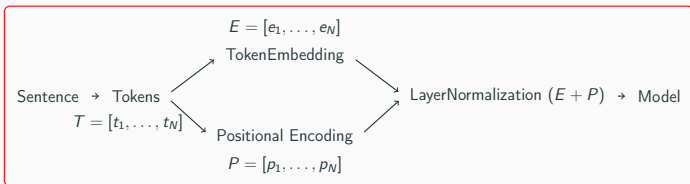
$$k \in \llbracket 1, N \rrbracket$$

$$p_k = (f_k(1), f_k(2), \dots, f_k(d_E))$$

$f_k(i)$: position de chaque
token dans la séquence

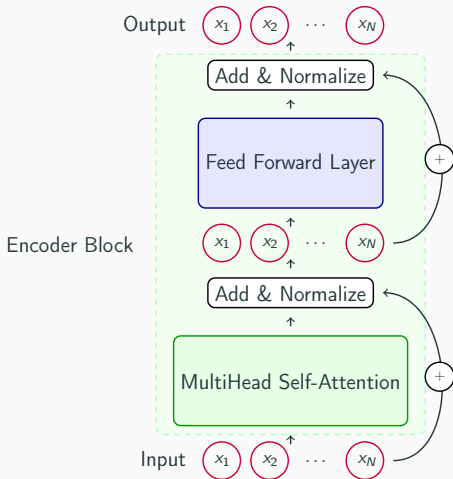
Token Embedding

$$t_k \longrightarrow \underbrace{(e_{k,0}, \dots, e_{k,d_E})}_{\text{dimension } d_E}$$

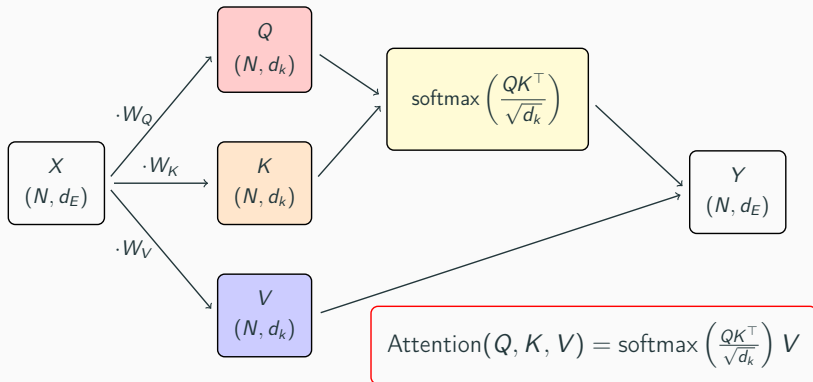


La partie Encodeur

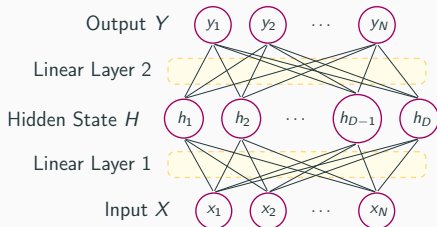
Schéma d'un block de la partie Encodeur de l'architecture Transformer :



Matrice d'attention



Le réseau Feed Forward



Couche linéaire :

$$X \mapsto X \cdot W^T + B$$

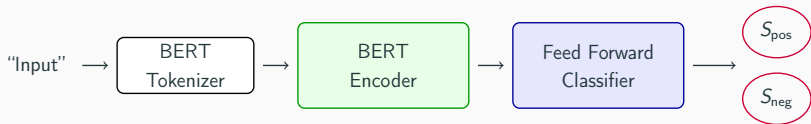
Application Personnelle : Objectifs & Rapport à la ville

- Objectif :
 - Classification de texte
 - Sentiment : Négatif \longleftrightarrow Positif
- Rapport à la ville :
 - Analyser les sentiments des habitants sur différents sujets
 - Peut servir à aider par exemple les mairies pour déterminer ce qui est le plus urgent

Le modèle BERT

- Architecture Transformer
- Plusieurs tailles de BERT (base, large)
- BERT base : $12 \times$ blocks encoder \rightarrow 112M paramètres
- BooksCorpus (800M words) and English Wikipedia (2,500M words)
- Publié vers fin 2018 par des chercheurs de Google

La structure du réseau de neurone utilisée



Les données et l'apprentissage

- **Données** : Twitter Sentiment140 (1.6 millions Tweets)
- **Train** : 50000, **Test** : 25000
- **Temps d'entraînement** : $\approx 5h$
- **Algorithme d'apprentissage** : Adam optimizer
(extension de la descente de gradient stochastique)
- **Fonction de Loss** : CrossEntropy

Les résultats

	My custom bert classifier			bertweet-base-sentiment-analysis		
	Positive	Neutral	Negative	Positive	Neutral	Negative
City (5784)	1940	<u>2006</u>	1838	1110	<u>2834</u>	1840
Cars (13104)	3272	4708	<u>5124</u>	1621	5466	<u>6017</u>
House (4681)	1501	<u>1675</u>	1505	720	<u>2297</u>	1664
Plants (1012)	197	371	<u>444</u>	81	430	<u>501</u>
Store (2087)	605	693	<u>789</u>	273	771	<u>1043</u>
Police (5834)	1496	<u>2323</u>	2015	482	<u>2797</u>	2555
Hospitals (4928)	1315	1634	<u>1979</u>	917	1765	<u>2246</u>

Ce que l'on a vu :

- Architecture Transformer
 - Le block d'encodeur
 - Les matrices d'attention
 - Les réseaux Feed Forward
- Application Personnelle

Annexes / Ouvertures :

- Détails du code python
- Adam optimizer
- La fonction de loss
- La Normalisation par couche (LayerNorm)
- La partie décodeur
- Comparaison avec le modèle GPT
- Le théorème d'approximation universel
- Bibliographie

Annexes : Code python

```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self, dim_in, dim_out):
        super().__init__()
        #
        self.layer = nn.Linear(dim_in, dim_out)

    def forward(self, x):
        return self.layer(x)
```

Annexes : Adam Optimizer

Annexes : Fonction de loss

Fonction Cross Entropy

Soit C_1, \dots, C_N N classes . Soit $x = (x_1, \dots, x_N)$ la sortie du modèle. Soit v l'indice de la vraie classe.

pour chaque $i \in \llbracket 1, n \rrbracket$,

$$E_i = -\log \left(\frac{e^{x_v}}{\sum_{k=1}^N e^{x_k}} \right)$$

Donc au final $CE(x, v) = \frac{1}{N} \sum_{i=1}^N E_i$

Annexes : LayerNormalization

Annexes : Partie décodeur de l'architecture transformer

Annexes : Comparaison avec le modèle GPT

Annexes : Le théorème d'approximation universel

Enoncé : Les transformateurs sont des approximateurs universels des fonctions continues de séquence à séquence sur un domaine compact.

Preuve :

Annexes : Bibliographie

- Pytorch documentation
- “Attention is all you need”, Google Research, 2017
- “BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding”, Google AI Language, 2018
- “Improving Language Understanding by Generative Pre-Training” OpenAI, 2018
- “ARE TRANSFORMERS UNIVERSAL APPROXIMATORS OF SEQUENCE-TO-SEQUENCE FUNCTIONS?”, Google Research, 2020