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

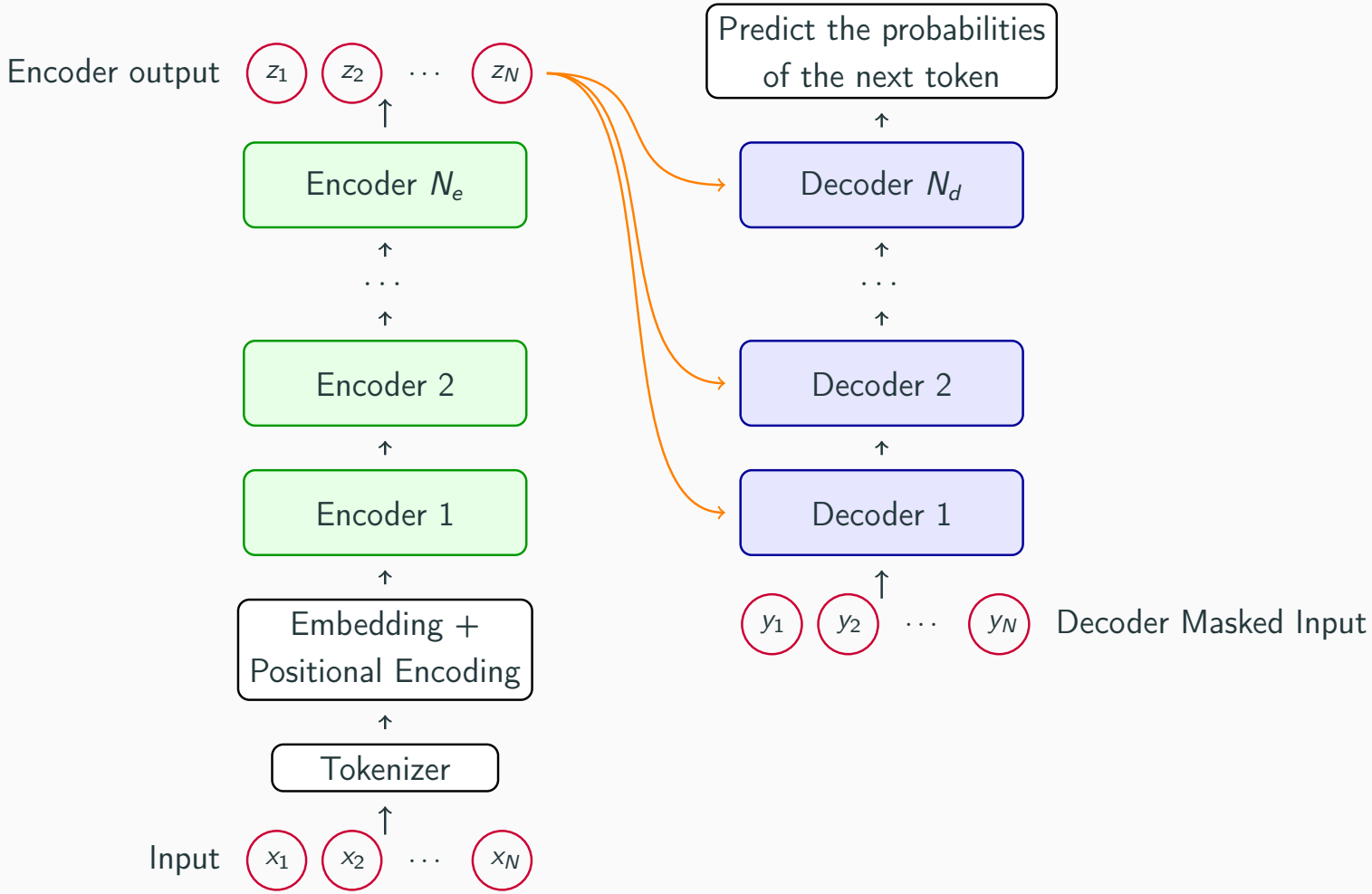
Cerisara Nathan, MPI

27 mai 2023

Sommaire

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 :
$[101, 15756, 6125, 2024, 2061, 4658, 999, 102, 0, \ldots, 0]$

[CLS] “neural” “networks” “are” “so” “cool” “ !” [SEP]

Ex2 :

SENTENCE : “Bonjour le monde !”

TOKENS :
$[101, 14753, 23099, 2099, 3393, 23117, 999, 102, 0, \ldots, 0]$

[CLS] “bon” “##jou” “##r” “le” “monde” “ !” [SEP]

Vectorisation du texte : Embeddings & Positional Encoding

Positional Encoding

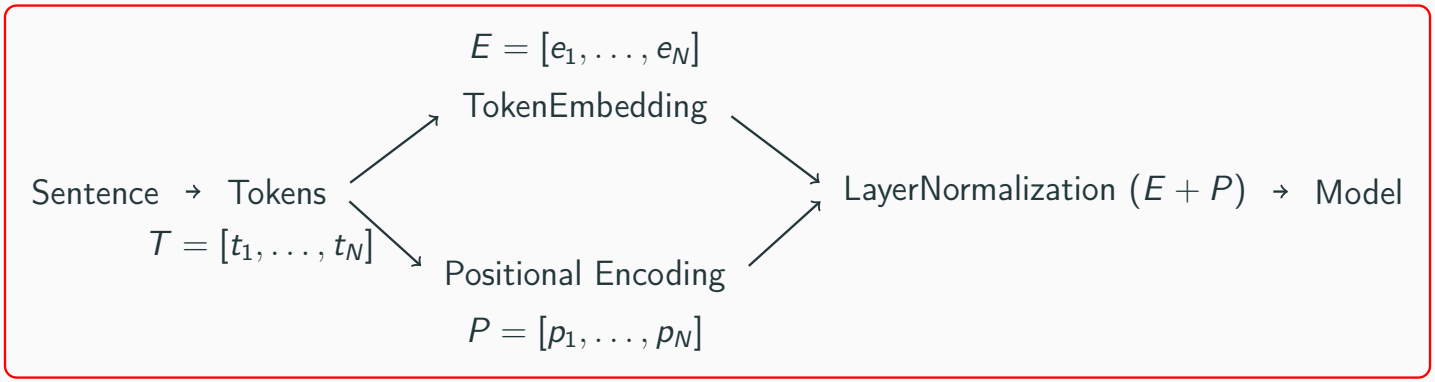
$$p_k = (s_0, c_1, \ldots, s_{2i}, c_{2i+1}, \ldots, s_N)$$

$$s_{2i} = \sin\left(k \cdot 10000^{-\frac{2i}{d_E}}\right)$$

$$c_{2i+1} = \cos\left(k \cdot 10000^{-\frac{2i+1}{d_E}}\right)$$

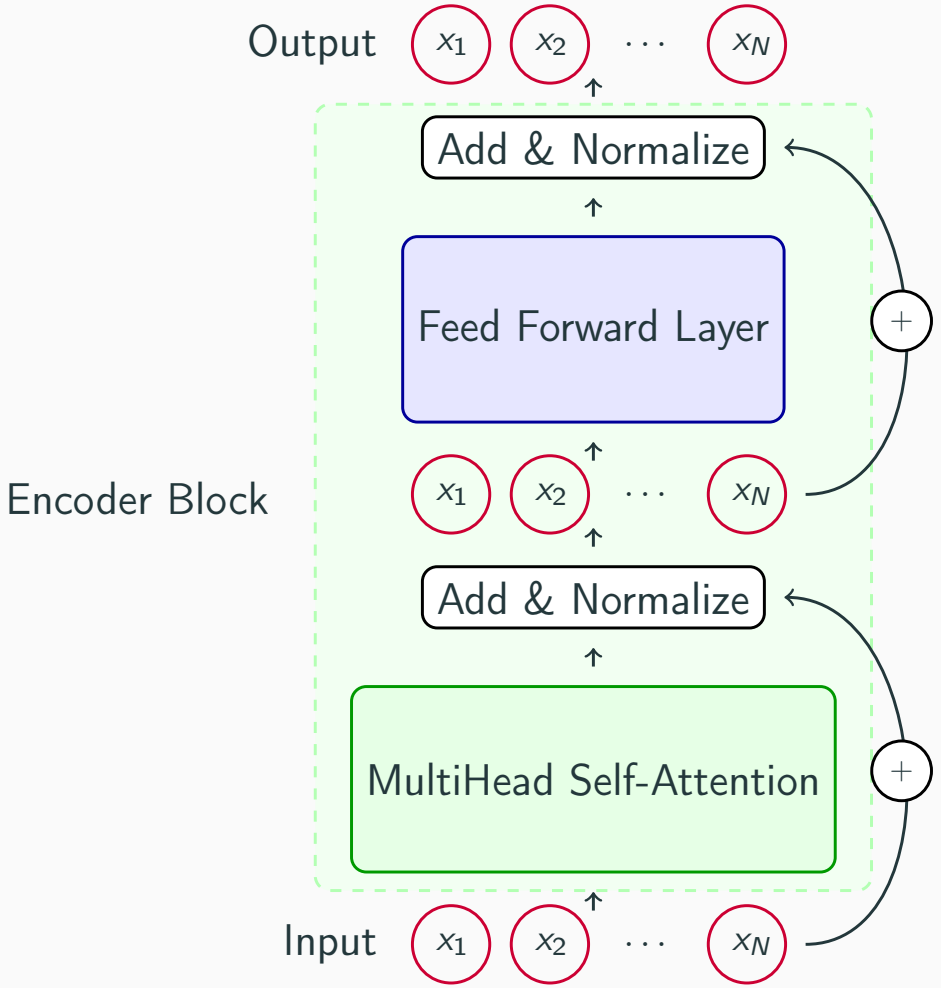
Token Embedding

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

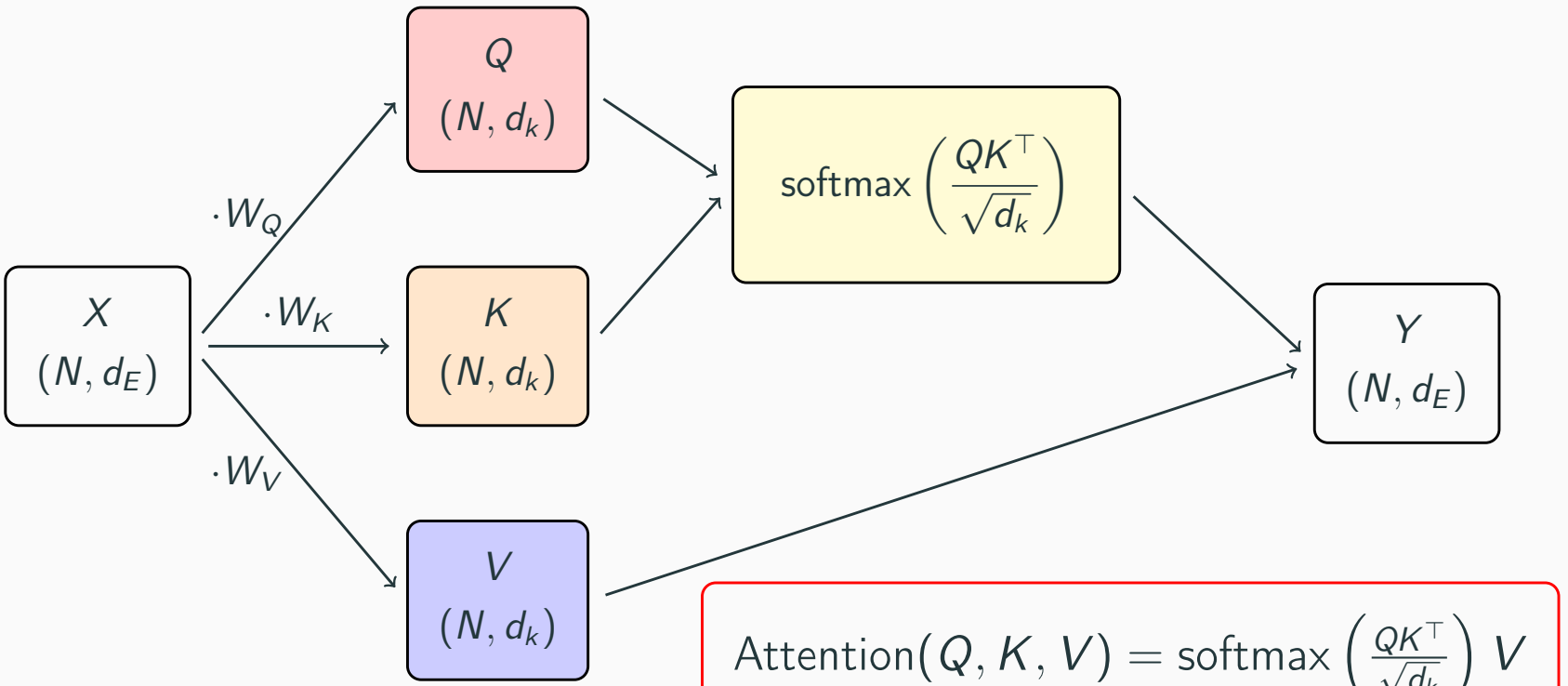


La partie Encodeur

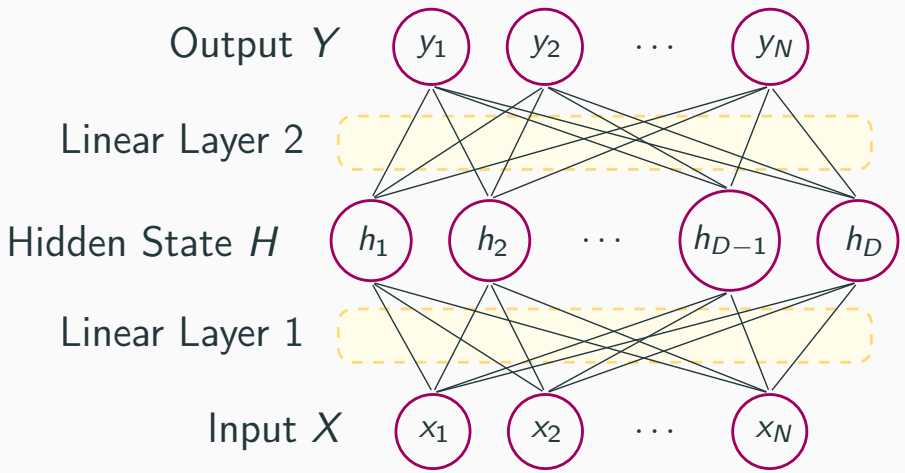
Schéma d'un block de la partie Encodeur de l'architecture Transformer :



Matrice d'attention



Le réseau Feed Forward



Couche linéaire :

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

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

Le modèle BERT

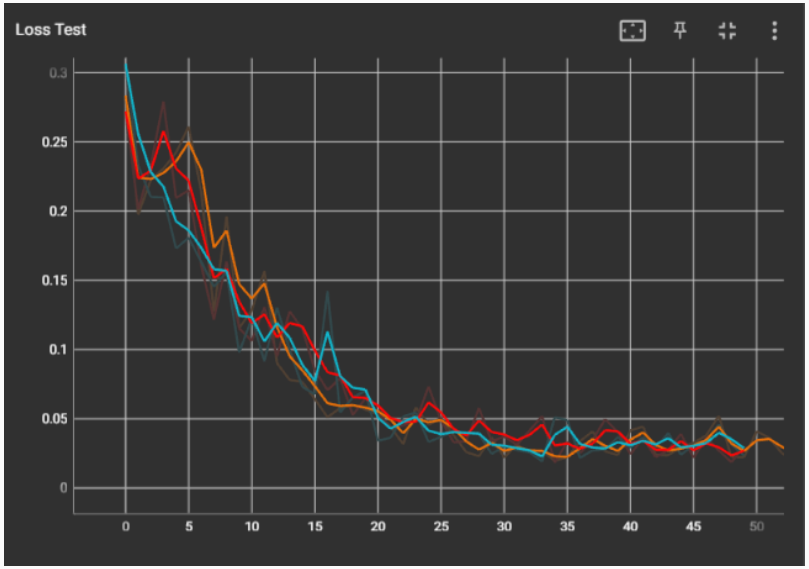
- Architecture Transformer
- Plusieurs tailles de BERT (base, large, ...)
- BERT base : 12$\times$ blocks encoder $\rightarrow$ 112M paramètres
- BooksCorpus (800M words) and English Wikipedia (2,500M words)
- Publié vers fin 2018 par des chercheurs de Google

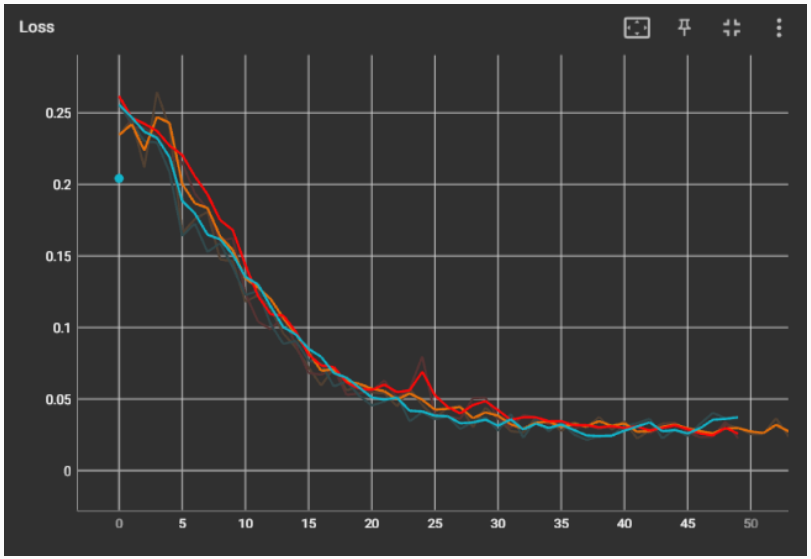
La structure du réseau de neurone utilisée

Les données et l'apprentissage

- **Données :** Standford Sentiment Dataset
- Train : 8544, Test : 2210
- **Temps d'entraînement :** $\approx$ 15min
- **Algorithme d'apprentissage :** Adam optimizer (extension de la descente de gradient stochastique)
- **Fonction de Loss :** MinSquaredError

Les résultats





Annexes

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

```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

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

- Les différents papiers
- “ARE TRANSFORMERS UNIVERSAL APPROXIMATORS OF SEQUENCE-TO-SEQUENCE FUNCTIONS ?”, Google Research 2020