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

Cerisara Nathan, MPI

SCEI: 10953

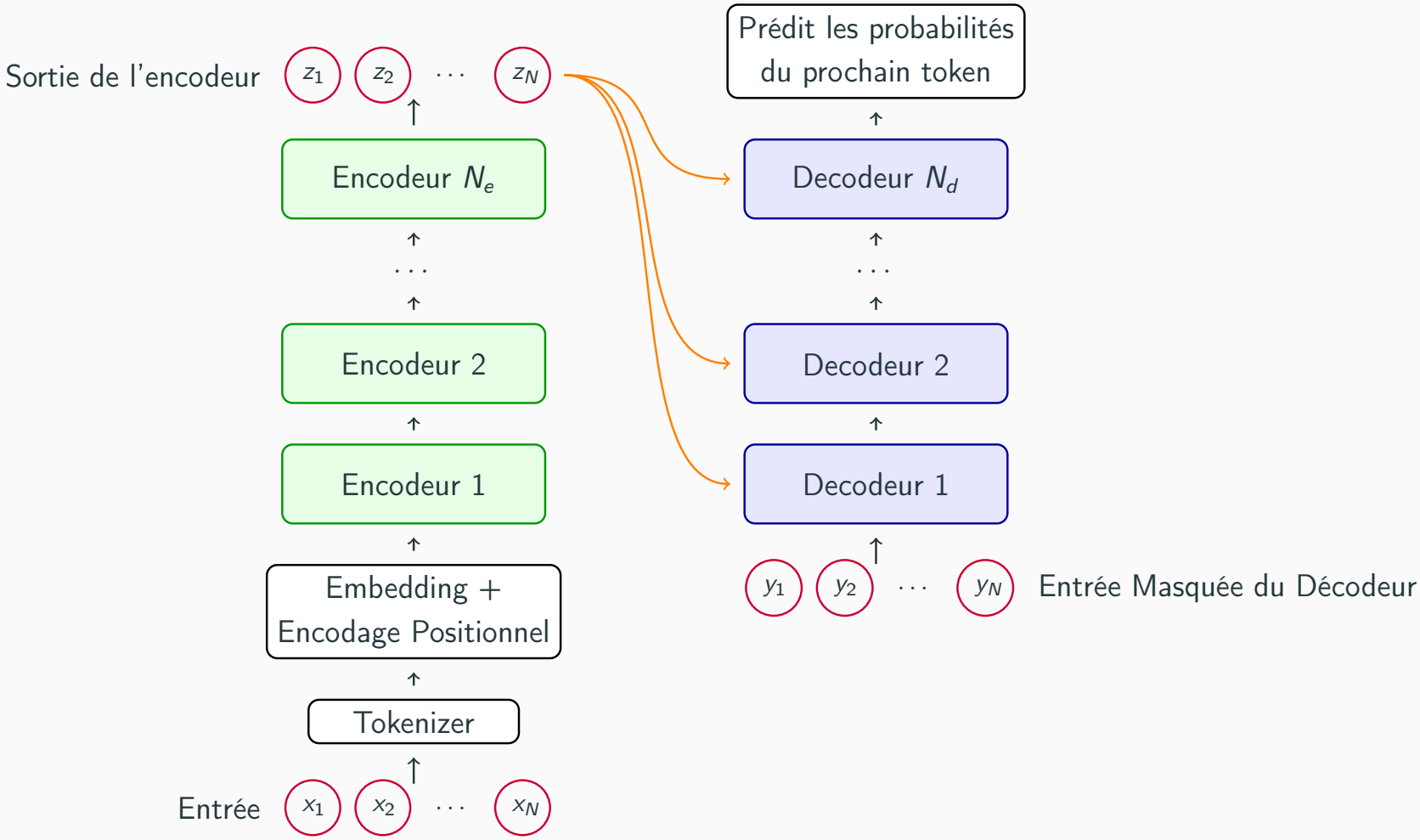
Plan de la présentation

1. Architecture Transformer
   - 1.1 Vectorisation du texte
   - 1.2 La partie Encodeur de l'architecture
   - 1.3 Les matrices d'Attention
   - 1.4 Le réseau Feed Forward

2. Application personnelle
   - 2.1 Objectif
   - 2.2 Le modèle BERT
   - 2.3 La structure du réseau de neurone utilisée
   - 2.4 Les données et l'apprentissage
   - 2.5 Les résultats

1. L'architecture Transformer

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



1.1 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]

1.1 Vectorisation du texte : Embeddings & Encodage Positionnel

Encodage Positionnel

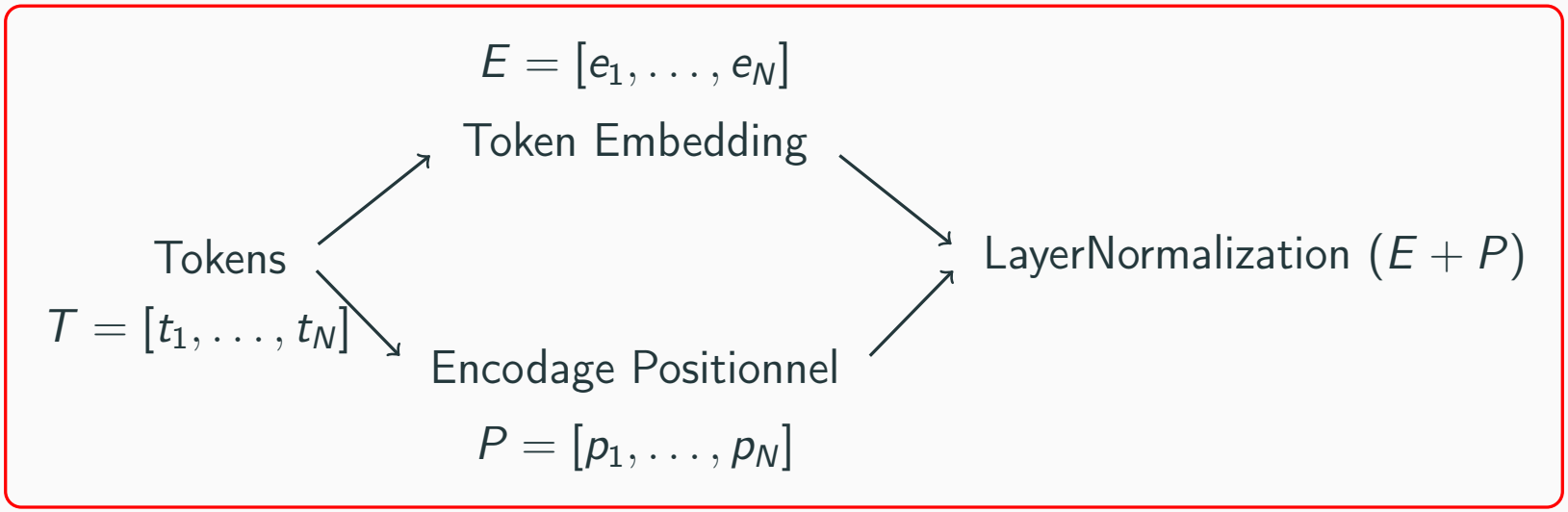
$k \in [\![1, N]\!]$

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

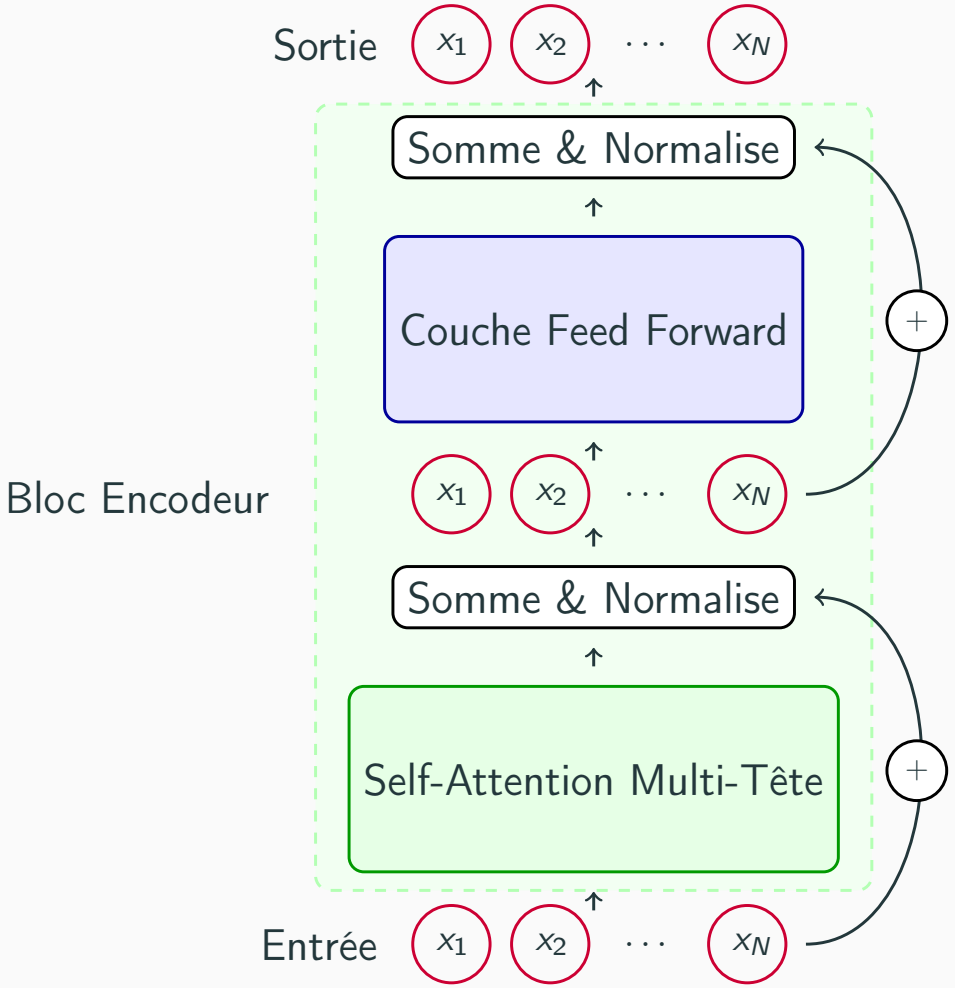
$f_k(i)$: position de chaque token dans la séquence

Token Embedding

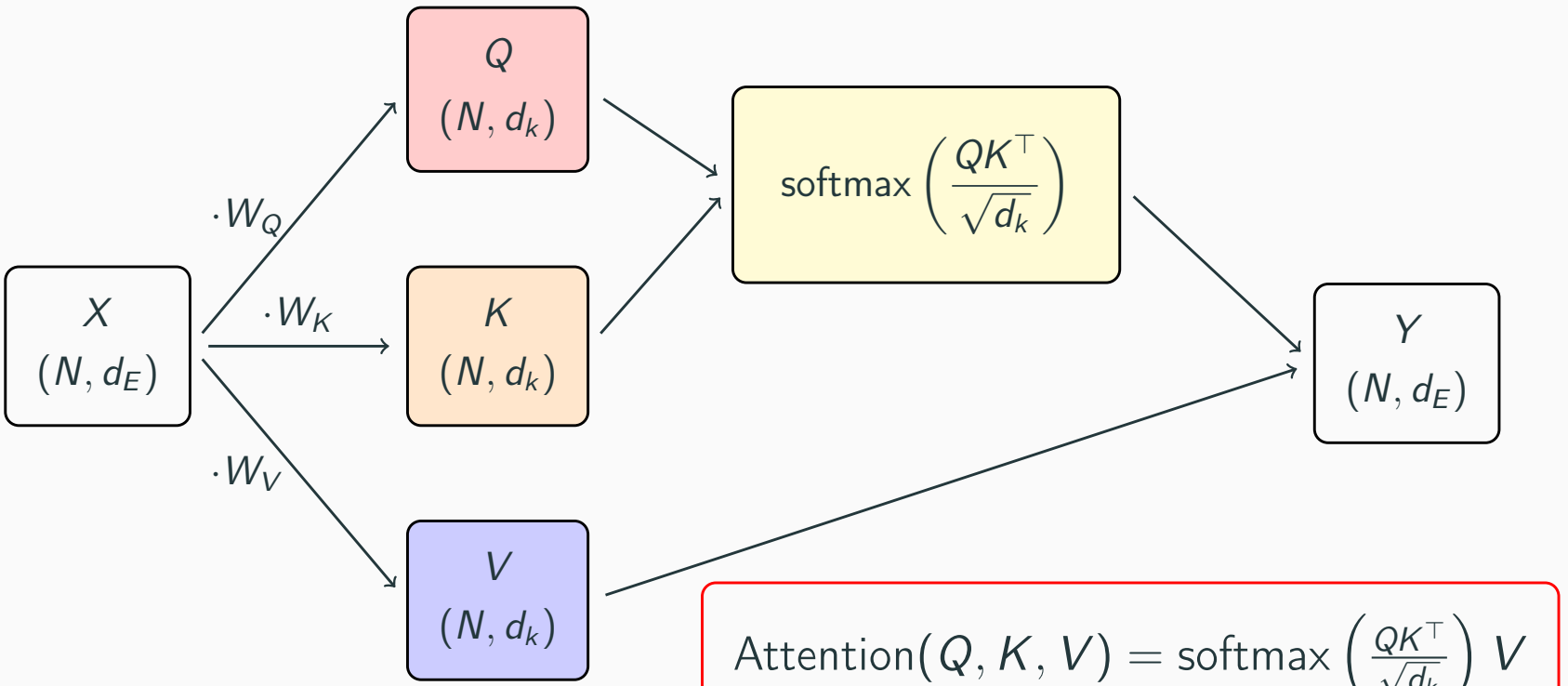
$$t_k \longrightarrow \underbrace{(e_{k,0}, \ldots, e_{k,d_E})}_{\text{dimension } d_E}$$



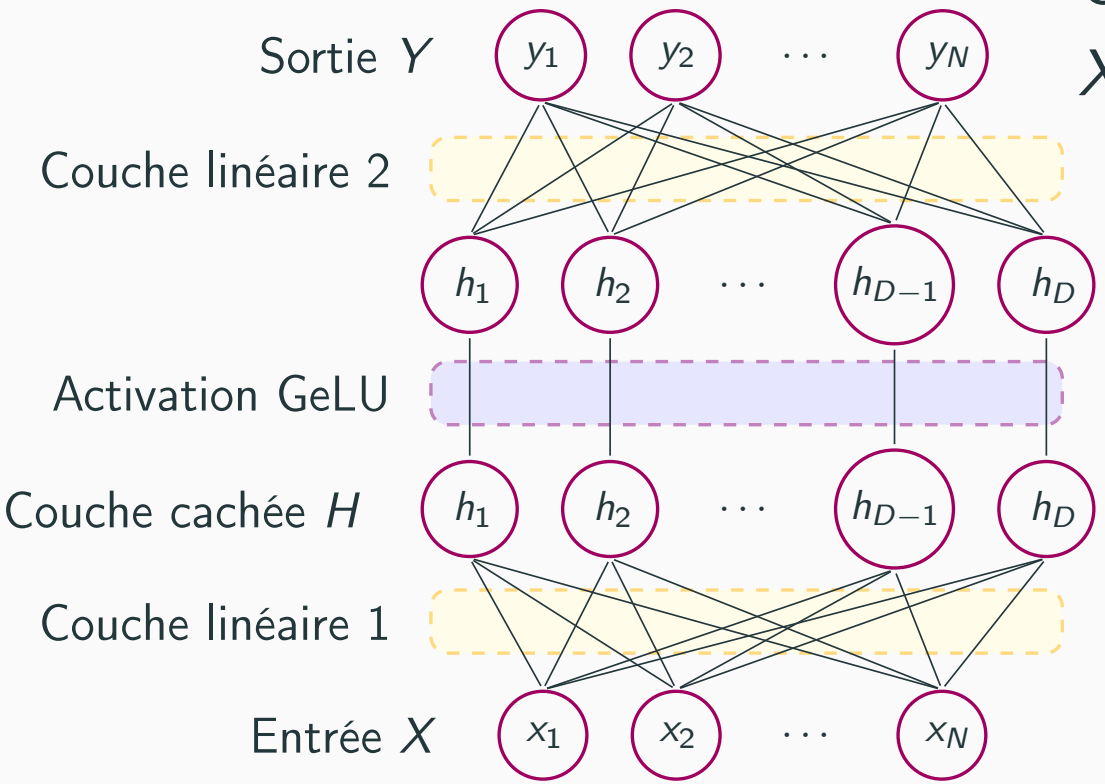
1.2 La partie Encodeur



1.3 Matrice d'attention



1.4 Le réseau Feed Forward



Couche linéaire :

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

2.1 Application Personnelle : Objectifs

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

2.2 Le modèle BERT

- Architecture Transformer (Encodeur seulement)
- 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

2.3 La structure du réseau de neurone utilisée

2.4 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

2.5 Les résultats

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

Twitter news dataset (https ://www.kaggle.com)

Conclusion

- L'architecture Transformer permet de construire des représentations pertinentes du langage naturel
- Plusieurs de tâches sont possibles : Classification, Génération, Traduction, ...
- classifieur de texte : analyser les sentiments d'un message
- Améliorations possibles : Meilleurs données d'entraînement, modèle plus large, ...

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 :

- Un peu de code python
- Adam optimizer
- La fonction de loss
- La Normalisation par couche (LayerNorm)
- Fonctions Softmax et GeLU
- La partie décodeur
- Comparaison avec le modèle GPT
- Bibliographie

Annexes : Code python - Classe Réseau de Neurone

Structure très simplifiée d'une classe réseau de neurone en pyton avec pytorch :

```
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 : Code python - Boucle d'entraînement

```
def train_model(self, epochs):
  dataloader = Dataloader(self.train_dataset,...)
  self.model.train()
  for epoch in range(epochs):
    for batch, data in dataloader:
      label = data['target']
      self.optimizer.zero_grad()
      output = self.model(
        ids= data['ids'],
        mask=data['mask'],
        token_type_ids=data['token_type_ids'])
      label = label.type_as(output)
      loss = self.loss_fn(output,label)
      loss.backward()
      self.optimizer.step()
```

Annexes : Adam Optimizer

Descente de gradient stochastique

Différences avec Descente de Gradient :

- Batch de données **vs** Dataset entier
- Plus efficace en terme de calculs
- Ne se bloque pas forcément dans un minimum local
- Plus flexible pour le taux d'apprentissage

Adam : Adaptive Moment Estimation

- pour chaque itération : Moyennes pondérées du gradient sur l'historique récent
- Taux d'apprentissage adaptatif (individuellement pour chaque paramètre)
- Correction de biais
- Mise à jour des paramètres

Annexes : Fonction de loss

Fonction Cross Entropy

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

pour chaque $i \in [\![1, n]\!]$,

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

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

Annexes : LayerNormalization

Entrée : Tensor X de dimensions : (batch_size, sequence_length, embedding_size)

Opérations :

- La moyenne $E(X)$ et la Variance $V(X)$ (sur la dimension de l'embedding de l'entrée)
- Normalisation :

$$N_X = \frac{X - E(X)}{\sqrt{V(X)}}$$

- Mise à l'échelle et décalage :

$$N_x \cdot \alpha + \beta$$

$$\text{LayerNorm(X)} = \frac{X - E(X)}{\sqrt{V(X) + \varepsilon}} \cdot \alpha + \beta$$

Annexes : Fonction Softmax et GeLU

Fonction Softmax :

Pour $z = (z_1, \ldots, z_N)$

$$\text{Softmax}(z) = \left(\frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}} \right)_i$$

Pour $i \in [\![1, N]\!]$

ReLU et GeLU :

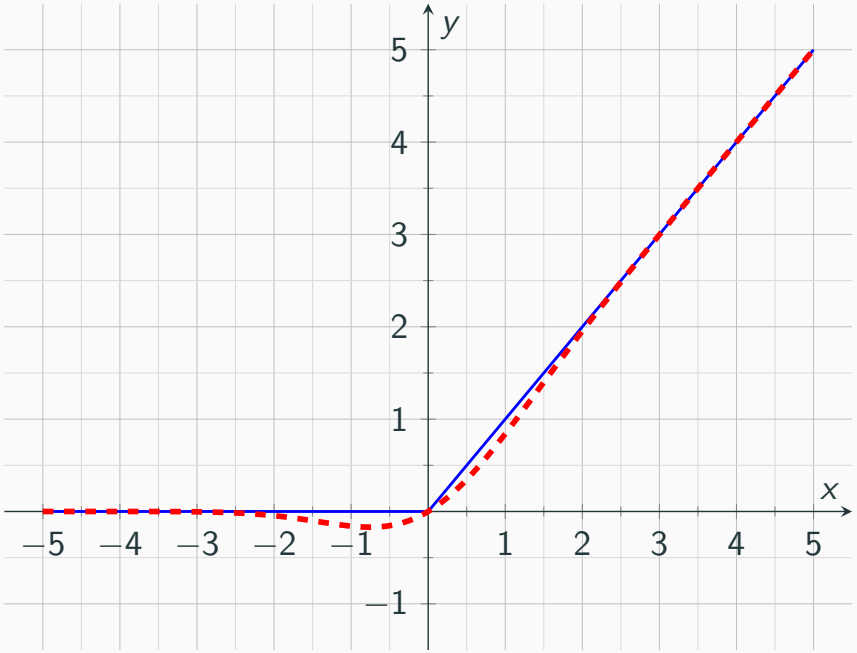
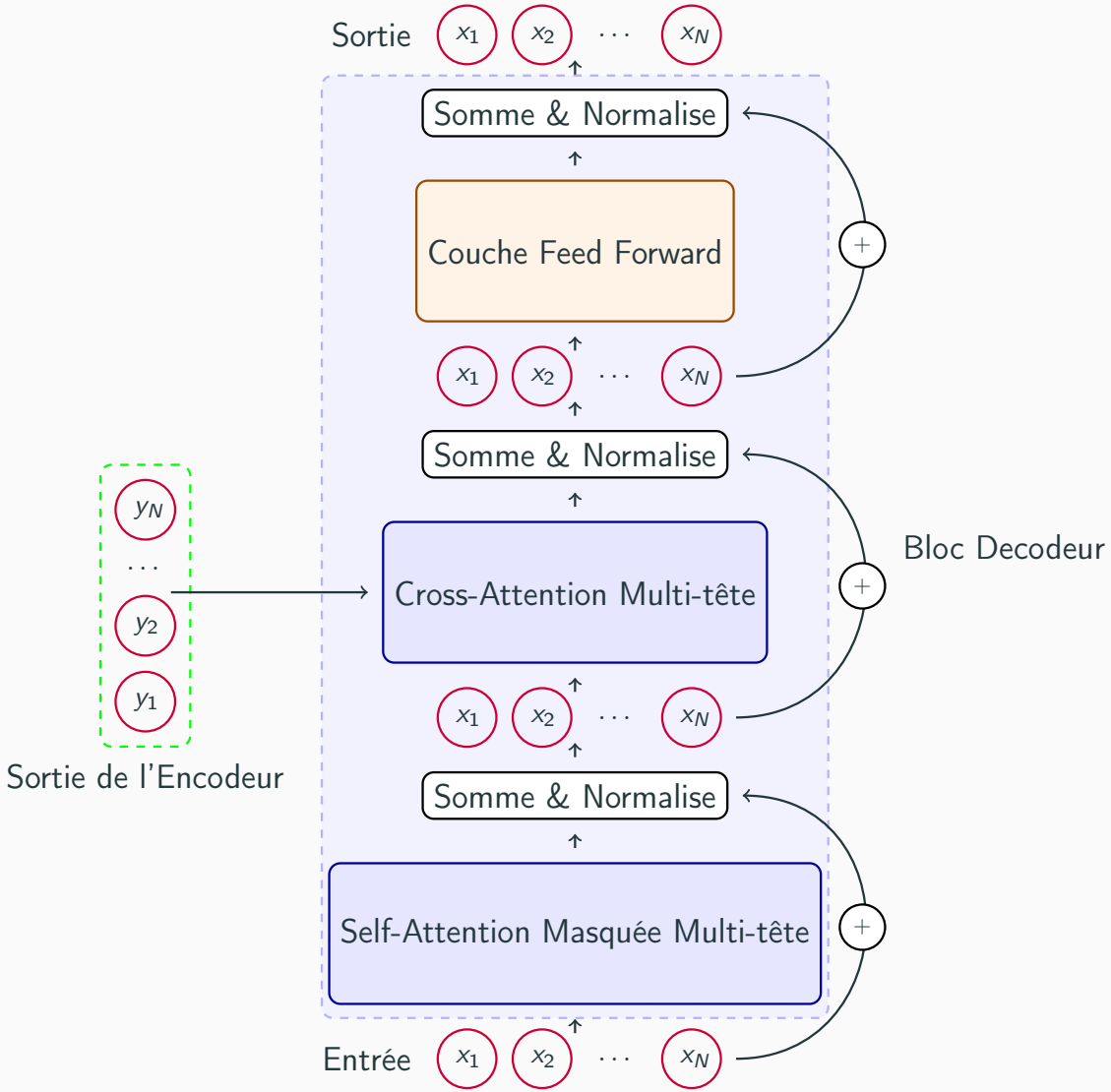


Figure 1: Fonctions ReLU et GeLU

Annexes : Partie décodeur de l'architecture transformer



Annexes : Comparaison avec le modèle GPT

Architecture :

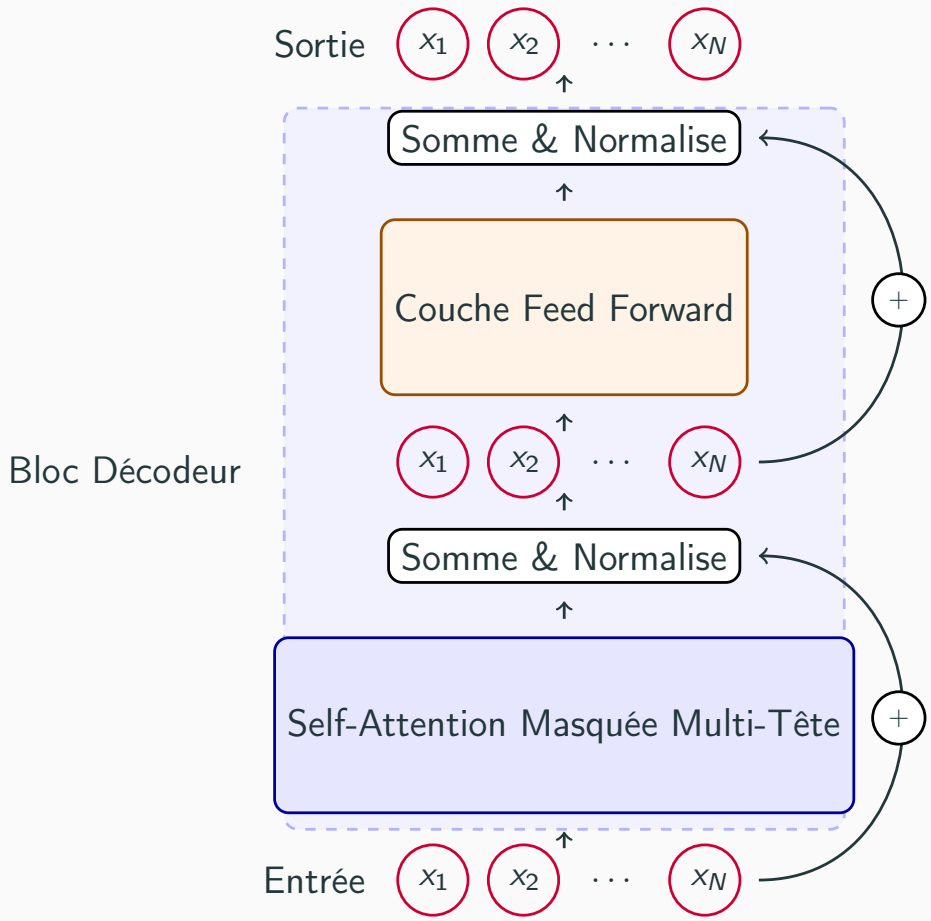
- BERT : Encodeur seulement
- GPT : Decodeur seulement

Contexte :

- BERT : Bidirectionnel
- GPT : A gauche seulement

Utilisation :

- BERT : Classification, Traduction
- GPT : Génération de texte



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
- “The Illustrated GPT-2 (Visualizing Transformer Language Models)” (https ://jalammar.github.io/illustrated-gpt2/)