Intro aux Transformers

28/11/25
Simon Coste

I Intro 1: Historique

Tout part du papier "Attention is all you need" 2017, Google 180K cit.
Premier modèle transf: BERT, RoBERT, Camembert → classif
→ analyse de sentiments
Ensuite : Generative Pretrained Transformer
↳ pas à BERT qui fait classif.
Aujourd'hui, modèles open source : LLAMA 4, DEEPSEEK, OLMO 3, MISTRAL
Rq : OLMO est full open source : poids, training. Vivre tech report savoir le coup.
Ceci sont des modèles de langages (LLM).
En 2021 on se rend compte que transformer peut remplacer les convolutions ds les modèles d'image "An image is worth 16x16 words" ce qui a conduit aux ViT (Vision Transformers). C'est bcp + fort aujourd'hui les modèles image gen n'utilisent plus les convs, seulement le transformer.
C'est un peu vrai aussi pour la classif

II Tokens

Dans les transformers les données sont des suites de tokens.
Un transformer est une (famille de) fonctions paramétrées $\mathbb{R}^{T \times d} \longrightarrow \mathbb{R}^{T \times d}$
où T = nb de tokens et d = dim du modèle. Dc on manipule des suites (de longueur T) de tokens de dim d.
⚠ d est fixé par contre T peut être amené à changer.

L'idée (qui vient du langage) est que 1 token = 1 unité sémantique.
Ex : langage : syllabe
vision : patch image 8x8
audio : morceau de spectrogramme

2) Tokenisation

Procédé transformant une suite de caractères en une suite de tokens.
Nb de caractères : ~ 200 Nb de mots : 500K (conjugaison inclus) mais en vrai juste 20K. Une lettre = trop petit, pas d'inf sémantique et pas de fusion.
Donc il faut choisir comment décomposer les mots en tokens.
Ex le site "Tiktokenizer" recense les tokenizers utilisés en pratique, le + populaire est Byte-Pair Encoding (BPE)
En général il y a 100K tokens → V = taille du vocabulaire
Ex : math + ematique. Même si "en" "ati" "que" existent dans la base !
Ex : les nombres c'est relou. Si on met tous les entiers jusque 1000 comment modéliser 10000 ? Il n'y a pas une unique façon de décomposer. 100␣00 ? 10␣000 ?
Rq : Historiquement c'était basé sur linguistique. En fait on s'est éloigné et automatisé le process. Par ex BPE prend les paires de lettres les + courantes, puis les triplets etc. C'est data-driven.

Une fois que la tokenization est faite, une phrase tokenisée (X) est de dimension $T \times 1$ ↑ numéro du token dans le dictionnaire.

Ça n'a pas de sens de mettre les entiers pour coder les tokens.

On préfère les encoder avec des vecteurs via $i \sim e_i$ ↑ base canonique de $\mathbb{R}^V$

Du coup X est de dimension $T \times V$. Trop gros.

Du coup on projette ces vecteurs sur un espace de dim $d \ll V$

Ici d est la "model dimension"

Ex : OLMo 3 : $d = 5120$.

3) Embedding

Cette projection se fait via une matrice de embedding / projection P de taille $V \times d$. ↑ indep de T !

et renvoie $x = XP \sim [T, d]$. C'est l'input du Transformer

Rq : $X = \begin{bmatrix} x_1 \\ \vdots \\ x_T \end{bmatrix}_T$ (V) et $x = \begin{bmatrix} \quad \end{bmatrix}_T$ (d)

Rq : P s'obtient via... learning. En général on le réapprend pour chaque application. C'est dû au fait que le vocabulaire ne contient pas "que" du texte mais aussi des infos telles que "ceci est une question de l'utilisateur" ou "ceci est du code latex". Et on va souvent rajouter des tokens au cours du temps. Du coup P va changer.

~~Ceci étant fait la pipeline est terminée~~

4) Decoding

Ceci étant dit la pipeline est la suivante :

$$\text{phrase} \xrightarrow[\text{token}]{} X \xrightarrow[\text{embedding}]{} x \xrightarrow[\text{transformer}]{} f(x) = o$$

On fait quoi de l'output o ?

Ex : On entraîne un classifier $\varphi : [T, d] \to \mathbb{R}$ (quelle langue pour ce texte ?)

$\varphi : [T, d] \to [T, 1]$ (est-ce qu'il y a un mot que l'on cherche (ou pas).)

C'était ce que faisait BERT

Ex : On veut faire de la traduction. Pour cela on réinjecte o dans $[T, V]$ puis on convertit les tokens en texte.

En général on entraîne un fonction de classif $\varphi : \mathbb{R}^d \to \Delta_V$ qui à un vecteur de $\mathbb{R}^d$ donne une proba sur $\{1 \dots V\}$.

Ex : Next-word prediction.

On a un output $o = (o_1 \dots o_T)$, $o_i \in \mathbb{R}^d$

On entraîne un classifier $\varphi : \mathbb{R}^d \to \mathbb{R}^V$ dont le job est que

$$\varphi(o_i) \approx X_{i+1}$$

Ainsi on espère que $\varphi(o_T)$ sera... X_{T+1} !

Voilà pour le concept général. Il reste à dire : c'est quoi le transformer f ???

III L'architecture des transformer

$f = f_L \circ \dots \circ f_1$ où L = nb de couches & f_i = bloc de transformer.

Ex : $L = 120$ pour GPT-4, $L = 70$ pr LLAMA 4

Le point clé est que les f_i aient des mécanismes d'attention. Il y en a plusieurs.

1) Self-attention

On a en input $x \sim [T, d]$ des matrices $Q, K, V \sim [d, d]$

et on pose
- $q = xQ$ → query
- $k = xK$ → key
- $v = xV$ → value

On définit une matrice de score $S = qk^T \sim [T, T]$ (gros !)

$S_{ij} = \langle q_i, k_j \rangle$

Matrice d'attention qui joue le rôle d'une matrice de transit° / Markov

$$A_{ij} = \frac{e^{S_{ij}}{\sum_{k=1}^{T} e^{S_{ik}}}$$ (en gros $A = e^S$ normalisée sur les lignes)

& les lignes sont des probas.

Output : $o = AV$

tel que $(Av)_i = \sum_{j=1}^{T} A_{ij} v_j$

Rq : cela s'écrit de façon condensée :

$x \mapsto \text{softmax}(xQK^T x)xV$ appelé $SATTN(x)$ (self)

Rq : Le + important c'est S : qui dit comment les tokens i et j interagissent entre eux. On pourrait apprendre S directement, mais elle doit dépendre de x

On choisit que $S = qk^T = xQK^T x$. On pourrait apprendre QK^T direct ~~mais c'est de l'info~~ ?

2) Cross-Attention

$$XATTN(x) = \text{softmax}(xQ^T K_y^T)yV$$

par ex x = FR y = EN

ou x = prompt y = image

3) Multi head Attention

$$MHATTN(x) = (SATTN_i(x)]_i .$$

01/12/25

Rappel : Un transformer est une fonction $f : \mathbb{R}^{Td} \to \mathbb{R}^{Td}$ T = nb tokens, d = dim notable

$f = f_L \circ \dots \circ f_1$ où chaque f_i est un bloc de l'architecture dit "bloc transformer"

chaque bloc f_i est lui même composé de couches, typiquement

f_i = "MLP" ∘ "Attention"

↳ s'appelle aussi Feed forward Network (ffn ou ffd)

la couche de self-attention est paramétrisée par des matrices $Q, K, V \in [d, d]$

et se résume en $x \mapsto \text{softmax}(xQK^T x)xV$.

Quelle interprétation ?

La matrice de score $S_{ij} = \langle q_i, k_j \rangle$. + c'est grand, + le token q_i donne de l'importance à la key k_j.

Ex : dictionnaire où key = "mots", value = "clef"

Supposons une query "mangées". Il n'est pas tel quel dans le dico.
Mais il y a "manger", "mangé". Donc on a 2 keys pertinentes pour la query.
Puis on va en extraire, combiner les values correspondantes.
Historiquement en info, base de données, ce sont des pbs/notions classiques.

Ensuite la matrice d'attention est $P_{ij} = \frac{e^{S_{ij}}}{\sum_{k=1}^{T} e^{S_{ik}}}$

telle que $\underbrace{\text{softmax}\left(\frac{\overbrace{xQK^T x}^{S}}{}\right)}_{P} \underbrace{xV}_{v} = Pv$, $v = xV$

Rq : On fait le choix que K, q, v sont linéaires en x. C'est pas obligé et des versions non linéaires existent. Mais le linéaire est suffisant en pratique.

Il y a bcp de variantes, et en pratique on utilise bcp Multi Head Attention

Def : On a des matrices Q^i K^i $V^i \sim [d, \frac{d}{H}]$ pour $i = 1..H$ ← nb de heads
On définit des $q^i = xQ^i$ k^i, v^i etc
output $o^i = \text{softmax}(xQ^i K^{iT} x) xV^i$
la couche renvoie la concaténation $(o^1, .., o^H) \sim [T, d]$

Ex : Group-Query attention : idem mais avec $K^i \equiv K$ $V^i \equiv V$
seuls les Q^i sont différents.
Par ex Key Value commun pour toutes les questions liées à la structure grammaticale
mais plein de Q^i différents : qui est le sujet, le verbe etc.

4) Complexité de l'architecture

Combien de calcul ? Évaluer la fonction c'est évaluer la matrice d'attention P, donc $\Theta(T^2)$ car il faut appliquer la softmax à une matrice T^2. Bcp de gens essaient de faire des couches d'attention qui coûtent moins que T^2.

En mémoire c'est un peu pareil T^2 mais + facile à améliorer

Ex : linear attention   Objectif : réduire les flops.

Rappelons que $P_{ij} = \frac{e^{\langle q_i, k_j \rangle}}{\sum_{\ell} e^{\langle q_i, k_\ell \rangle}}$. Puisque $e^{\langle q_i, k_j \rangle}$ est une similarité, on pourrait prendre n'importe quelle autre fonction de similarité $\varphi(q_i, k_j)$, $\varphi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$.

Idée à la RKHS : $\varphi(x, y) = \langle \sigma(x), \sigma(y) \rangle$ ← en pratique ReLU...

Alors $P_{ij} = \frac{\langle \sigma(q_i)\, \sigma(k_j) \rangle}{\sum_{\ell} \langle \sigma(q_i)\, \sigma(k_\ell) \rangle} = \frac{\langle \sigma(q_i), \sigma(k_j) \rangle}{\langle \sigma(q_i), \underbrace{\sum_{\ell} \sigma(k_\ell)}_{:= z} \rangle}$

Donc $P = \{\{ \text{diag} \frac{1}{\langle \sigma(q_i), z \rangle} \cdot \underset{[T,d]}{\sigma(q)} \underbrace{\sigma(k)^T x V}_{(d,d)}$

O, bien que P soit TxT ici on peut tricher : en fait on ne manipule que des matrices dxd ou dxT

Donc ici on a une complexité en Td.

Est-ce qu'on a perdu ?

Papiers : · l'expressivité du réseau et la n?
· + dur à entraîner (avant l'inférence)
· mélanger les 2 pour gagner en temps en vision (à l'inférence)
· Les non linear attention [illegible]

5) Le bloc transformer

On rappelle que $f_i = FFN \circ ATTN$

Le FFN est une fonction qui s'applique individuellement à chaque token, càd $FFN(o) = (\varphi(o_1), \dots, \varphi(o_T))$ où φ est un NN

Rq: En pratique φ est un ResNet ie un réseau résiduel

Il y a aussi des layers de normalization (layer-norm, ada norm, rms norm) qui sont des fcts $RMS(x) = \frac{x}{|x|}\gamma$ où γ est un paramètre à apprendre.

root mean square au lieu de $\frac{x}{|x|^2}$ → en pratique $\frac{x}{\sqrt{|x|^2+\varepsilon}}$

C'est une question de savoir où les mettre. Par ex:

X → RMS → ATTN → RMS → FFN → output.

- la normalisation permet que le réseau voit toujours des choses du même ordre cela empêche des explosions et stabilise le système.
- le résidu dit qu'en fait o est un mouvement de déplacement un peu comme la discrétisation d'une ODE $x \to x+o$.

6) Le FFN

a) La tradition: FFN est un MLP : $\varphi(x) = \underset{[dd']}{A_2}\, \sigma(\underset{[d'd]}{A_1} x)$ ($\mathbb{R}^d \to \mathbb{R}^{d'}$)

Multi layer Perceptron.

Utilisé à l'époque (GPT-2)

b) Gated Activation (SWIGLU
switch gated linear unit

$SWIGLU(x) = \underset{[dd']}{A_3}(\underset{[d'd]}{A_2} x \odot \sigma(\underset{[d'd]}{A_1} x))$ ← c'est quadratique en x!
← typiquement $d' = 4d$ ou $3d$ ↑ MLP ratio
$\sigma(x) = \frac{xe^x}{1+e^x}$ "swish"

c) Mixture of Experts (Jordan 85)

On a une famille de MLP / SWIGLU "experts" $\varphi_1 \dots \varphi_E$

On a un classifier "router" $I : x \mapsto I(x) \in \{1 \dots E\}$ ← 8 ou 32

puis $MOE(x) = \varphi_{I(x)}(x)$

Rq: Énorme impact des 3 dernières années.
Par ex la taille a augmenté énormément (×E) mais à l'inférence c'est un + vite.

Ex: Mistral 8×7B avec 8 experts. Deepseek: 32

Rq: c'est super dur à entraîner : si un expert devient relativement trop fort, il va éclipser les autres. Il y a plein de recettes, par ex on entraîne avec une contrainte que $KL(P_I, unif) \sim$ petit.

7) Flash attention   Tri Dao 2022

C'est une implémentation du layer attention avec un gain x10, sur GPU.

Il observe que sur GPU il y a 2 mémoires : HBM et Static RAM

Par défaut le GPU stocke tout sur HBM (~40 Go) qui a un BandWidth pas si High de 1-2 Tb/s → lent à écrire/lire

la SRAM est ~100 Kb mais a BW de 8 To/s.

Process : le GPU lit les données de la HBM, l'envoie à la SRAM
la SRAM envoie vite au processeur, qui calcule. Puis ça redescend
à la HBM (lent)

A priori la self attention demande de manipuler $P \sim [T, T]$ donc des opérations d'ordre T^2 → T^2 accès à la HBM → cher.

L'idée serait de découper cette matrice en petits morceaux ~~de taille T~~

on gagne un facteur $\frac{d^2}{M}$ où M = taille de la SRAM. En gros d'ordre $\sim \frac{1}{10}$.