

# CHAT GPT (LLM) Large Language Model ou auto-regressive

on a un début de phrase  $x_1, x_2, \dots, x_n$   $\leftarrow$  token

le modèle donne la distrib de proba de  $x_{n+1}$

On sait que GPT3 possède 150 Mld paramètres  
(nbs flottants 32 bits)

↪ 680 Go | une RTX 4090 a 24 Go de VRAM  
il faut 9 A100 a 80 Go de VRAM

Llama (Meta) 7B, 13B, 30B, 70B pr entraîner Llama  
il faut 2048 GPU A100 sur une période de 14j

Mistral 7B

Hugging Face (réseau social de l'open source en IA)  
Ils ont entraîné Palm 134j avec = 50 A100

Rai: production en énergie de la France par an  
≈ consommation des data center du monde.

Intérêt de la compression:

- \* énergie / écologie
- \* argent / économie
- \* latence / fps réel
- \* RGPD / données personnelles

Soit  $F$  le réseau VGG BN 16

avec 16 couches  $(f_1, \dots, f_{16})$

les 13 premières couches sont donnée par

$$f_i : \underset{\substack{\text{tenseur} \\ \text{en particulier} \\ \text{des features maps}}}{X} \mapsto \underset{\substack{\text{batch norm} \\ \uparrow \\ \text{convolution}}}{[BN(w * X + b)]}$$

Rappel: la batch normalisation normalise les features

$$BN: X \rightarrow \gamma \frac{x - \mu}{\sigma} + \beta$$

mesuré  
obtenus par SGD

$$E[BN(X)] = \beta \in \mathbb{R}^2$$

$$E[X] = \mu \in \mathbb{R}^2$$

$$V[BN(X)] = \gamma^2 \in \mathbb{R}^2$$

$$V[X] = \sigma^2 \in \mathbb{R}^2$$

A l'entraînement la BN mesure  $\mu$  et  $\sigma$  et les enregistre

Algo BN:  $\mu_c, \sigma_c \leftarrow x \in \mathbb{R}^{n \times h \times w \times c}$  nbs de canaux

$$\text{Signifie "courant"} \quad \mu \leftarrow 0,9 \mu + 0,1 \mu_c$$

$$\sigma \leftarrow 0,9 \sigma + 0,1 \sigma_c$$

$$y \leftarrow \gamma \frac{x - \mu_c}{\sigma_c} + \beta$$

A l'inference, la BN utilise les statistiques issues de l'entraînement. Algo BN:  $y \leftarrow \gamma \frac{x - \mu}{\sigma} + \beta$

Rappel: que peut-on dire d'une composée de transformato affines ?  
C'est une transformato affine.

Donc, on va pouvoir folder l'opération de BN dans les couches affines consécutives ou précédentes

Ex:  $f: X \mapsto BN(\omega X + b)$

écrivez  $f$  de la façon suivante:  
 $f = g: X \mapsto \tilde{\omega}X + \tilde{b}$

$$f(X) = g\left(\frac{\omega X + b - \mu}{\sigma}\right) + \beta$$

$$= \left(\frac{\gamma}{\sigma}\omega\right)X + \left(\gamma\frac{b-\mu}{\sigma} + \beta\right)$$

$$= \tilde{\omega}X + \tilde{b}$$

- Rq:
1. On ne peut pas tjr folder la BN (DenseNet & NasNet)
  2. lorsqu'on peut folder une BN  $\Leftrightarrow$  elle ne sert qu'à l'entraînement
  3. En général, on gagne entre 0,5% et 4% de paramètres
  4.  $f = g$

entrée de 100

$$\rightarrow O \xrightarrow{\omega X + b} O \xrightarrow{BN} O$$

$\omega X + b$   
 $100 \times 100 = 10000$

$\xrightarrow{N, \sigma, \gamma, \beta}$

$\xrightarrow{1, 100, 100, 100, 100}$

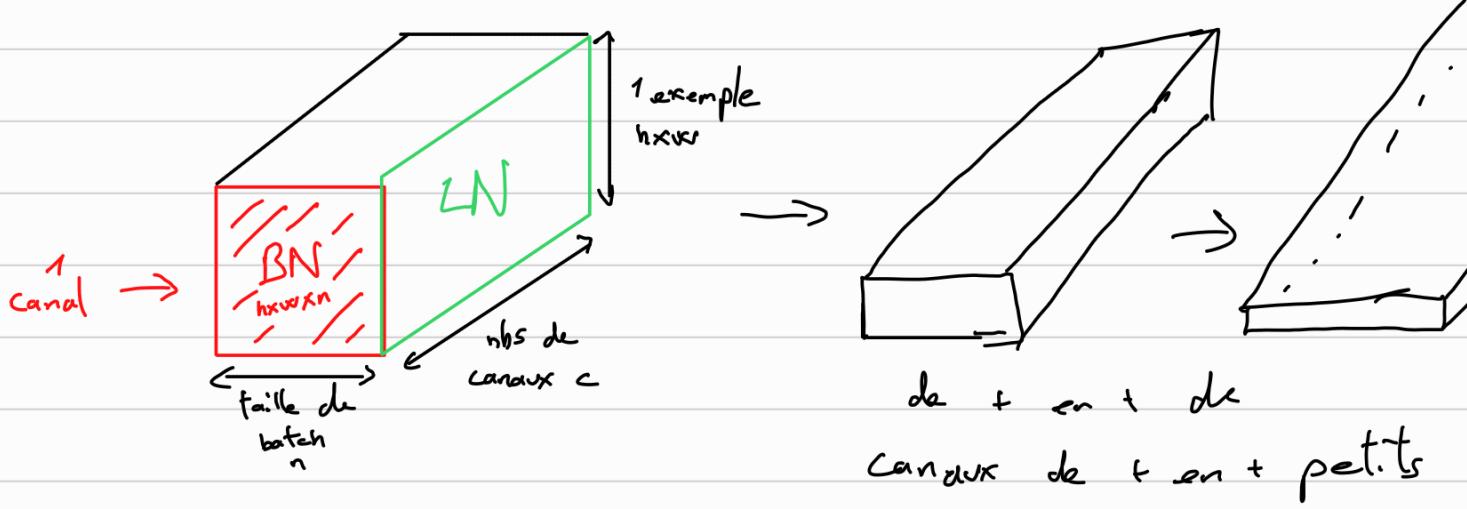
$\xrightarrow{\tilde{\omega}X + \tilde{b}}$

$100 \times 100 + 100$

On a vu une première méthode de compression, le Folding.  
Les BN sont en train de mourir avec CNN (ResNet, EfficientNet)  
et sont remplacés par les LN.

On a 2 diff entre LN et BN: \* l'axe de normalisation

\* les LN n'ont pas de mode "inference"



Folding Partiel de la LN (Layer Normalization)

$$LN(wx + b) = \gamma \frac{wx + b - M(wx+b)}{\sigma(wx+b)} + \beta$$

avec  $X \in \mathbb{R}^v$ ,  $w \in \mathbb{R}^{v \times v}$  et  $M = \begin{pmatrix} x_1 & \dots & x_v \\ x_1 & \dots & x_v \end{pmatrix} \in \mathbb{R}^{v \times v}$

$$(M y)_i = \sum_{j=1}^v x_j M_{ij} = \frac{1}{v} \sum_{j=1}^v x_j = E[y]$$

$$LN(wx+b) = \gamma \frac{(Id - M)x + b(Id - M)}{\sigma(wx+b)} + \beta$$

$$\tilde{LN}(\tilde{w}x + \tilde{b}) = \gamma \frac{\tilde{w}x + \tilde{b}}{\sigma(\tilde{w}x + \tilde{b})} + \beta$$

On a gagné le calcul de la moyenne et simplifié le calcul de l'écart-type

Comment aller plus loin ?

- \* réduire les matrices des poids = pruning  $\rightarrow$  retirer des calculs / poids
- \* Transformers

Pruning: On distingue 2 paradigmes en pruning. Le pruning par similarité et le pruning par magnitude. On a donc des critères de pruning. Par simplicité, travaillons sur une couche cachée

$$f : X \mapsto W_2 \sigma(W_1 X)$$

On ne touche pas aux entrées ni aux sorties du réseau. Ainsi, on ne prunera que les neurones de sortie de  $W_1$  (ou d'entrée de  $W_2$ ).

un neurone  $n_i$  de sortie de  $W_1$  est caractérisé par la ligne  $i$  de  $W_1$ .

Par défaut la similarité entre deux neurones  $n_i$  et  $n_j$  est donnée par

$$S(i, j) = \| (W_1)_i : - (W_1)_j \|_2$$

Par défaut, la magnitude est donnée par

$$M(i) = \| (W_1)_i \|_2$$

Si on veut retirer un neurone :

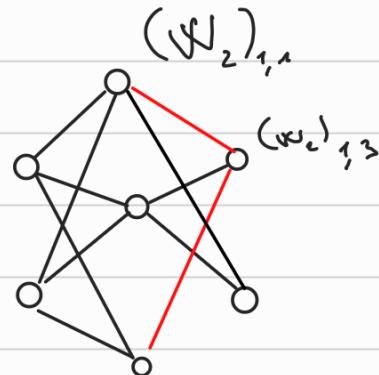
**Similarité** :  $\arg \min_{i \neq i'} S(i, i')$

1. on garde le neurone  $n_i$  pour faire les calculs de  $n_i$  et de  $n_{i'}$

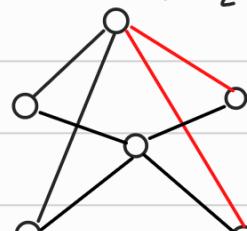
2. et on met à jour la couche suivante avec

$$(\tilde{W}_2)_{i, j} = (W_2)_{i, j} + (W_2)_{i, i'}$$

$$(\tilde{W}_2)_{i, j} = (W_2)_{i, j}$$



$$(W_2)_{i, i'} + (W_2)_{i, i'}$$



$\tilde{W}_2 = W_2$  privé  
de la ligne  
 $n_i$

En prunant, on a retiré autant de poids que d'entrées dans  $W_1$  plus le nb de sorties de  $W_2$ .

- rq:
1. Il existe des critères plus sophistiqués
  2. empiriquement, c'est souvent moins efficace que le pruning par magnitude.

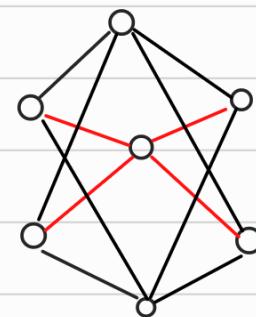
\* Magnitude :  $\arg \min_{\mathbf{C}} M(\mathbf{C})$

$$\tilde{\mathbf{X}}_1 = \mathbf{W}_1 \text{ privé des neurones i}$$

Pour simuler le pruning par magnitude

$$\mathbf{W}_1 \mathbf{X} = \begin{pmatrix} w_{11} & \dots & w \\ 0 & \dots & 0 \\ w & \dots & w \end{pmatrix} \mathbf{X} = \begin{pmatrix} n_1 \\ 0 \\ n_i \end{pmatrix} = \mathbf{Y}$$

$$\mathbf{W}_2 \mathbf{Y} = \left( \begin{array}{c} \left( \begin{array}{c} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{array} \right) \\ \text{ième} \end{array} \right) \left( \begin{array}{c} n_1 \\ \vdots \\ 0 \\ \vdots \\ n_i \end{array} \right) \text{ième}$$



$$\mathbf{W}_1 \in \mathbb{R}^{3 \times 2} \text{ et } \mathbf{W}_2 \in \mathbb{R}^{2 \times 3}$$

$$\tilde{\mathbf{W}}_2 = \mathbf{W}_2 \text{ privé de sa } i^{\text{ème}} \text{ colonne} \in \mathbb{R}^{2 \times 2}$$

Rq: 1. On a retiré des neurones de ce cas, on parle de pruning structuré

2. Dropout est un pruning non-structuré pour l'entraînement (non-statique)

3. Comment choisir le nbr de neurones à retirer

4. Si on a une grande perte de précision on peut ré-entraîner le modèle (fine-tuning)

Le pruning non-structuré ne modifie pas le graphe mais met certains poids à zéro.

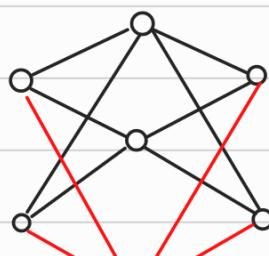
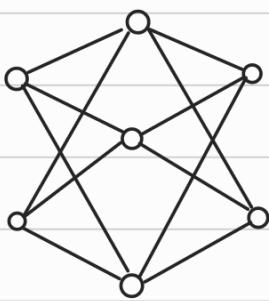
Le gain en inference vient des algorithmes de multiplication sparses (matrices plein de zéro)

Rq: \* sur CPU good

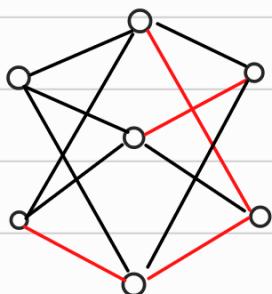
GPU de mieux en mieux mais pas encore top

\* le structuré est beaucoup plus difficile que le non-structuré à nbr de paramètres égal.

\* Attention le structure est simple à exploiter mais n'est pas très exploitable (hardware)



structuré



m - structuré

Autres critères d'importance:

- \*  $M(i) = \text{DC}(i)$  par SVD
- \*  $M(i) = \|\nabla_{w_i} F\|$  par gradient

paranthèses explicabilité:

l'attribution cherche à identifier les éléments importants parmi les inputs  
 $\Rightarrow$  Méthode la plus connue GradCam

meilleur critère ojrd: Integrated Gradients

Pruning et entraînement

\* fine-tuning : train  $\rightarrow$  prune  $\rightarrow$  train

\* IP (lottery ticket 2018) train  $\rightarrow$  prune  $\rightarrow$  init  $\rightarrow$  train  $\rightarrow$  prune

\* prune at init init  $\rightarrow$  prune  $\rightarrow$  train