

Série S3 : Maîtrise de l'Architecture et Optimisation

AI & LLMs

Université Ibn Tofail

Exercice 1 : La Dynamique de la "Température"

Sujet : Contrôle de la créativité lors de l'inférence.

Contexte Théorique : La Température (T)

Dans le cours, nous avons vu la fonction **Softmax** standard qui transforme les scores bruts (logits) en probabilités. Cependant, lors de la génération de texte (Inférence), nous voulons parfois moduler la "créativité" du modèle sans réentraîner les poids.

Pour cela, on divise les logits (z) par un hyperparamètre T (Température) avant d'appliquer l'exponentielle. La formule devient :

$$P_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

- Si $T \rightarrow 0$: Le modèle devient "froid" et déterministe (il choisit toujours le mot le plus probable).
- Si $T \rightarrow \infty$: La distribution s'aplatit, rendant tous les mots équiprobables (aléatoire total).

Supposons que le modèle hésite entre deux mots pour terminer la phrase "Le ciel est...". Les Logits (scores bruts z) sortant du réseau sont :

- "Bleu" : $z_1 = 4.0$
- "Gris" : $z_2 = 2.0$

Questions :

1. **Cas Standard** ($T = 1$) : Calculez les probabilités $P(\text{"Bleu"})$ et $P(\text{"Gris"})$ avec la formule standard. Le modèle est-il confiant ?
2. **Cas "Créatif"** ($T = 2$) : Calculez les nouvelles probabilités en divisant d'abord les logits par 2. L'écart de probabilité entre "Bleu" et "Gris" a-t-il augmenté ou diminué ?
3. **Cas "Déterministe"** ($T = 0.5$) : Calculez les probabilités en divisant les logits par 0.5 (ce qui revient à les multiplier par 2). Que constatez-vous sur la probabilité du mot dominant ?

Exercice 2 : La Chaîne de Rétropropagation (Deep Learning)

Sujet : Application concrète de la "Chain Rule".

Contexte Théorique : Le Flux de Gradient

Dans un réseau profond, nous ne pouvons pas calculer directement l'impact d'un poids de la première couche sur l'erreur finale, car il y a des couches intermédiaires. Nous utilisons la **Règle de la Chaîne** (Chain Rule) : pour savoir comment A affecte C , nous regardons

comment A affecte B , puis comment B affecte C , et nous multiplions ces impacts.

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \text{Sortie}} \times \frac{\partial \text{Sortie}}{\partial \text{Caché}} \times \frac{\partial \text{Caché}}{\partial w_1}$$

Considérons un mini-réseau linéaire : Entrée (x) $\xrightarrow{\times w_1}$ Caché (h) $\xrightarrow{\times w_2}$ Prédiction ($pred$)

Données : $x = 2$, $w_1 = 3$ (donc $h = 6$), $w_2 = 0.5$ (donc $pred = 3$). La cible réelle est $y = 4$. L'erreur est $E = (pred - y)^2$. Le gradient de l'erreur par rapport à la prédiction est $\frac{\partial E}{\partial pred} = 2(pred - y)$.

Questions :

1. Calculez la valeur numérique du gradient local de l'erreur : $\frac{\partial E}{\partial pred}$.
2. Nous voulons mettre à jour le poids profond w_1 . Appliquez la formule de la chaîne pour trouver $\frac{\partial E}{\partial w_1}$.
3. **Analyse (Vanishing Gradient) :** Si le poids w_2 avait été très petit (ex : 0.001) au lieu de 0.5, quel impact cela aurait-il eu sur le gradient reçu par w_1 ? Cela aiderait-il ou empêcherait-il w_1 d'apprendre ?

Exercice 3 : La Complexité Quadratique

Sujet : Les limites physiques de la fenêtre contextuelle.

Contexte Théorique : Complexité $O(N^2)$

Le mécanisme de Self-Attention est puissant car chaque token "regarde" tous les autres tokens de la séquence. Cela signifie que pour une séquence de longueur N , nous devons calculer une matrice de scores de taille $N \times N$. Si on double la longueur du texte ($2N$), le nombre de calculs est multiplié par 4 ($2N \times 2N = 4N^2$). On appelle cela une complexité **quadratique**.

Questions :

1. Si la fenêtre de contexte est de $N = 4,000$ tokens, combien de scores d'attention le modèle calcule-t-il (taille de la matrice) ?
2. Si l'on passe à une fenêtre de $N = 8,000$ tokens, par combien est multiplié le nombre de calculs ?
3. Supposons que chaque score occupe 2 octets en mémoire vidéo (VRAM). Quelle quantité de mémoire (en Mégaoctets) est nécessaire *juste* pour stocker cette matrice pour une séquence de $N = 32,000$ tokens ?

(Rappel : 1 Mégaoctet $\approx 10^6$ octets).

Exercice 4 : L'Encodage Positionnel (Positional Encoding)

Sujet : Comment le Transformer gère-t-il l'ordre des mots sans récurrence ?

Contexte Théorique : Le problème de l'Invariance

Contrairement aux RNNs qui lisent phrase mot après mot (séquentiel), le Transformer lit toute la phrase d'un coup (parallèle). Cependant, cette puissance a un défaut majeur : les opérations mathématiques du Transformer (Produit Scalaire dans l'Attention) sont **symétriques**.

Exemple : La similarité entre "Jean" et "Mange" est la même, que Jean soit le sujet (avant le verbe) ou non. Pour le modèle brut, la phrase :

"Jean aime Marie"

est mathématiquement identique à :

"Marie aime Jean"

C'est ce qu'on appelle un "Sac de Mots" (Bag of Words).

La Solution : Pour réintroduire l'ordre, nous ajoutons un vecteur spécifique à la position (appelé P) au vecteur du mot (appelé E).

$$\vec{V}_{\text{Entrée}} = \vec{E}_{\text{Mot}} + \vec{P}_{\text{Position}}$$

Ainsi, le vecteur final contient à la fois le sens du mot ET sa position.

Questions :

- Analyse conceptuelle :** Supposons une architecture Transformer sans Encodage Positionnel. Nous avons deux phrases :
 - Phrase A : "Le chat chasse la souris"
 - Phrase B : "La souris chasse le chat"
 Dans la Phrase A, le vecteur du mot "chat" est-il strictement identique au vecteur du mot "chat" dans la Phrase B ? Si oui, pourquoi est-ce un problème pour comprendre qui est le prédateur ?
- Application Mathématique :** Montrons comment l'addition vectorielle résout ce problème. Travaillons dans un espace simplifié à 2 dimensions.
 - Le vecteur sémantique (sens) du mot "Bob" est : $\vec{E}_{Bob} = [10, 10]$.
 - Le vecteur de la Position 1 (Sujet) est : $\vec{P}_1 = [1, 0]$.
 - Le vecteur de la Position 3 (Objet) est : $\vec{P}_3 = [0, 5]$.

Calculs :

- Calculez le vecteur final représentant "Bob" lorsqu'il est en début de phrase (Position 1) : $\vec{V}_{Bob_Sujet} = \vec{E}_{Bob} + \vec{P}_1$.
- Calculez le vecteur final représentant "Bob" lorsqu'il est en fin de phrase (Position 3) : $\vec{V}_{Bob_Objet} = \vec{E}_{Bob} + \vec{P}_3$.
- Les deux vecteurs résultants sont-ils identiques ? Expliquez en une phrase comment cela permet au mécanisme d'Attention de faire la différence entre "Bob qui agit" et "Bob qui subit".

Exercice 5 : Sur-apprentissage (Overfitting)

Sujet : Interpréter les courbes de Loss.

Contexte Théorique : Généralisation

L'objectif d'un LLM n'est pas de réciter ses données d'entraînement (mémoire), mais de comprendre les règles de la langue pour les appliquer à de nouveaux textes (généralisation).

- **Training Loss :** Erreur sur les données que le modèle connaît (doit descendre).
- **Validation Loss :** Erreur sur des données inconnues (le vrai test). Si elle remonte alors que la Training Loss descend, le modèle fait du "par cœur".

Vous entraînez un modèle et observez ces courbes :

- **Époque 10 :** Train Loss = 3.0 | Val Loss = 3.2
- **Époque 20 :** Train Loss = 2.1 | Val Loss = 2.3
- **Époque 30 :** Train Loss = 1.5 | Val Loss = 2.8
- **Époque 40 :** Train Loss = 0.8 | Val Loss = 4.5

Questions :

1. À quelle époque précise auriez-vous dû arrêter l'entraînement ("Early Stopping") ?
2. À l'époque 40, le modèle est-il intelligent ou stupide ? Expliquez ce qu'il est en train de faire concrètement avec les phrases d'entraînement pour avoir une Training Loss si basse (0.8) et une Val Loss si haute.