Optimisation d'hyperparamètres avec NOMAD pour le fine-tuning des grands modèles de langage

Approche et résultats

Septembre 2023–Février 2024

Sacha BENARROCH-LELONG et Christophe TRIBES

Table des matières

I. Introduction

Ce document vise à fournir une compréhension des expériences menées et des outils utilisés dans le cadre du projet Optimisation d'Hyperparamètres (*Hyperparameter Optimization*, abrégé HPO dans ce document) pour les grands modèles de langage (*Large Language Models*, abrégé LLM dans ce document) au GERAD. Il décrit le travail qui a été effectué pour la publication de notre article dans l'Edge Intelligence Workshop de la conférence AAAI24 [Tri+23].

I.1. Objectifs du projet

Tous les termes seront définis au fur et à mesure.

Nous réalisons l'*instruction-tuning* de la version à 7 milliards de paramètres de LLaMA 2 avec la technique de *Low-Rank Adaptation* (LoRA). Cette procédure fait intervenir plusieurs hyperparamètres, parmi lesquels nous essayons d'optimiser 4 avec `NOMAD`. Nous menons une étude comparative avec l'algorithme *Tree-structured Parzen Estimator* (TPE) implémenté dans le logiciel `NNI`.

I.2. Organisation du document

Nous commencerons par un tour d'horizon théorique pour décrire :

— le fonctionnement des grands modèles de langage comme LLaMA ;

— le principe de *transfer learning* dont découlent *fine-tuning* et *instruction tuning* ;

— la technique de *fine-tuning* LoRA.

Nous décrirons ensuite notre démarche expérimentale et évoquerons quelques conclusions que nous avons tirées de notre travail pour suggérer de futures pistes.

Lorsque c'est nécessaire, nous évoquons en début de section des prérequis théoriques qui sont nécessaires à comprendre pleinement les concepts, et supposés connus du lecteur.

II. Grands modèles de langage (LLMs)

Prérequis

Fonctionnement général d'un réseau de neurones.

II.1. TOKENS ET EMBEDDINGS

Une première étape nécessaire pour traiter du texte dans un réseau de neurones est de convertir ce texte en valeurs numériques. Nous donnons 2 définitions grossières.

Tokens Il s'agit de sous-ensembles atomiques d'une séquence de mots. La *tokenization* est le processus de découpage de la séquence en éléments : mots, propositions, découpage selon la ponctuation, etc.

Embeddings Il s'agit de la représentation numérique des tokens. L'idée général d'un embedding est de traduire n'importe quel token en un vecteur de taille constante $d \in \mathbb{N}^*$. Il existe plusieurs techniques pour leur calcul, mais une bonne technique doit traduire les liens sémantiques en des liens numériques.
Exemple : Soient f une fonction représentant un calcul d'*embeddings*. On veut par exemple :

- $f(\text{"roi"}) \simeq f(\text{"reine"})$: deux mots de sens proches sont proches dans l'espace des embeddings ;
- $f(\text{"reine"}) + f(\text{"homme"}) \simeq f(\text{"roi"})$: la combinaison sémantique se retrouve dans l'addition des embeddings.

Par $x \simeq y$, on entend $\|x - y\| < \varepsilon$ pour un certain ε suffisamment petit, puisque les embeddings sont des vecteurs.

II.2. MÉCANISME D'ATTENTION

La plupart des modèles de langage performants reposent aujourd'hui sur une architecture de réseaux de neurones appelée *Transformers*. Cette architecture a été introduite en 2017 [Vas+17] et repose essentiellement sur un mécanisme appelé *attention*.

L'attention permet à un réseau de neurones de prendre en compte les liens qui existent entre les mots au sein d'une phrase. La variante proposée dans l'introduction des Transformers est basée sur des calculs de produits scalaires et des transformations linéaires.

Matrices de poids Soient $n \in \mathbb{N}^*$ et $(x_i)_{i\in[n]}$ des embeddings de dimension $d \in \mathbb{N}^*$ réunis en colonnes dans la matrice $X \triangleq [x_1, ..., x_n] \in \mathbb{R}^{d\times n}$. Définissons 3 matrices :

- la matrice de poids des *clés* : $W_K \in \mathbb{R}^{d_k\times d}$;
- la matrice de poids des *requêtes* : $W_Q \in \mathbb{R}^{d_Q\times d}$;
- la matrice de poids des *valeurs* : $W_V \in \mathbb{R}^{d_V\times d}$,

où les nombres de lignes sont quelconques (non nuls). Posons ensuite :

$$\forall i \in [n], \quad k_i \triangleq W_K x_i, \quad q_i \triangleq W_Q x_i, \quad v_i \triangleq W_V x_i \ .$$

Scores d'attention L'objet du mécanisme d'attention est de calculer des scores évaluant l'intensité des liens entre les mots dont les *embeddings* sont contenus dans X. Ce calcul se fait en 3 étapes :

1. on applique à chaque *embedding* 3 transformations linéaires :

$$\forall i \in [n], \quad k_i \triangleq W_K x_i, \quad q_i \triangleq W_Q x_i, \quad v_i \triangleq W_V x_i \ .$$

   Pour $i, j \in [n]$, le produit scalaire $q_i^\top k_j$ doit fournir une mesure du lien entre les *embeddings* x_i et x_j. Si l'*embedding* est correctement calculé, ce lien est donc celui qui unit les mots traduits par x_i et x_j ;

2. pour chaque $i \in [n]$, le vecteur de ces produits scalaires est normalisé (par $\sqrt{d_K}$) puis transformé en vecteur de probabilités par la fonction $\mathrm{softmax}$ définie comme suit :

$$\begin{array}{rccc} \mathrm{softmax}: & \mathbb{R}^n & \to & \mathbb{R}^n \\ & t & \mapsto & \left[\frac{\exp(t_i)}{\sum_{k=1}^n \exp(t_k)}\right]_{1\leq i\leq n} \end{array} \ ;$$

3. on calcule les produits scalaires entre ces vecteurs de probabilités et les $(v_j)_{j\in[n]}$ pour obtenir le score final.

Si l'on note $K \triangleq W_K X, Q \triangleq W_Q X$ et $V \triangleq W_V X$, on a l'expression matricielle des scores d'attention :

$$\mathrm{Attention}(Q, K, V) = \mathbf{softmax}\left(\frac{Q^\top K}{\sqrt{d_K}}\right) V^\top \ .^{1\,2} \tag{1}$$

Multi-head attention Les auteurs de [Vas+17] proposent de regrouper $h \in \mathbb{N}^*$ mécanismes d'attention calculant leurs scores en parallèle, et les agrégeant à l'aide d'une dernière transformation linéaire W_O. Chaque mécanisme est appelé *tête d'attention* et possède ses propres matrices de poids.

Entraînement Dans un réseau de neurones contenant un mécanisme d'attention, la phase d'entraînement vise à déterminer les coefficients des matrices W_K, W_Q, W_V pour chaque tête d'attention, et des matrices W_O pour chaque multi-head attention.

II.3. TRANSFORMERS

Un Transformer est un réseau de neurones construit sur une architecture encodeur-décodeur utilisant le mécanisme d'attention. La structure globale est présentée en figure II.1. On observera surtout les trois blocs oranges correspondant à des modules d'attention. Nous allons décrire en détails les structures de l'encodeur (bloc gris à gauche de la figure) et du décodeur (bloc gris à droite), mais ce niveau de compréhension dépasse ce qui est nécessaire dans le cadre de notre projet. Ces deux paragraphes peuvent être négligés s'ils n'intéressent pas le lecteur.

1. $\mathbf{softmax}$ est la version matricielle de la fonction $\mathrm{softmax}$:

$$\begin{array}{rccc} \mathbf{softmax}: & \mathbb{R}^{n\times m} & \to & \mathbb{R}^{n\times m} \\ & X = [x_1, ..., x_m] & \mapsto & \begin{bmatrix} \mathrm{softmax}(x_1) & ... & \mathrm{softmax}(x_m) \end{bmatrix} \end{array} .$$

2. L'équation n'est volontairement pas exactement la même que dans [Vas+17], afin de rester cohérents avec les dimensions des matrices utilisées ici.

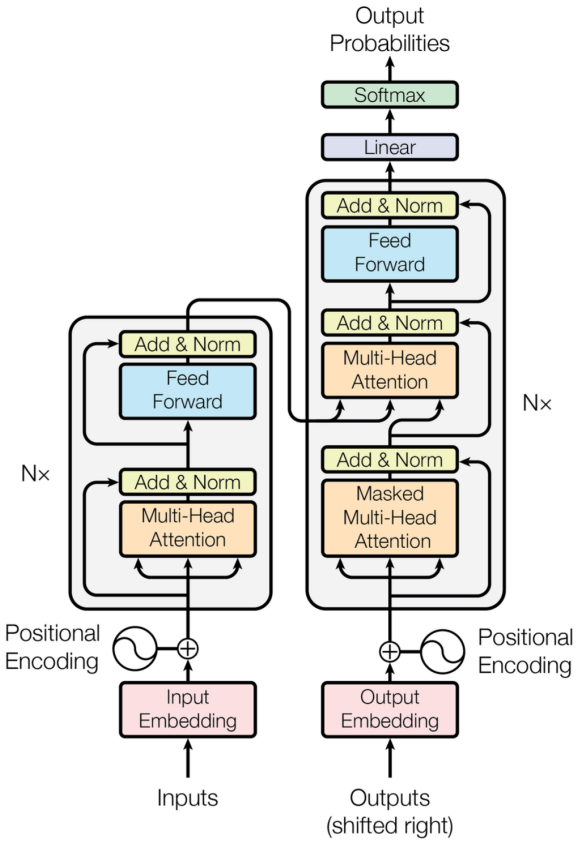

FIGURE II.1 – Architecure d'un Transformer. L'encodeur est représenté à gauche de la figure et le décodeur à droite. Figure tirée de [Vas+17].

L'encodeur La partie gauche de la figure II.1 représente l'encodeur. La phase pré-encodage reçoit les embeddings d'entrée et applique d'abord une technique de *positional encoding*. Nous ne la détaillerons pas ici, mais les auteurs y modifient les embeddings par l'ajout de termes judicieusement choisis afin de traduire l'importance de la position des mots dans la séquence. Ceci étant fait, les embeddings modifiés entrent dans l'encodeur et passent par un mécanisme de multi-head attention. La sortie est ensuite normalisée et transmise à un réseau classique de type *feed-forward*. Après nouvelle normalisation, le résultat est passé au décodeur. Pour la suite, notons Y ce résultat.

Le décodeur La partie droite de la figure II.1 représente le décodeur. Son action repose sur 2 mécanismes de *multi-head attention*.

1. Le premier est une *multi-head attention masquée*. Le décodeur doit pouvoir prédire les mots de façon séquentielle, ainsi lorsqu'il traite le $k^{\text{ème}}$ embedding de la séquence, les $(k-1)$ embeddings déjà prédits lui sont fournis en entrée (pour $k = 1$, l'embedding marquant le début de la séquence est passé). Cette première *multi-head attention* est appliquée sur ces $(k-1)$ embeddings. Cependant, lors de la phase d'entraînement du réseau, ce ne sont pas les embeddings déjà prédits qui sont passés à cet endroit, mais l'intégralité de la *ground truth* (prédiction attendue) du jeu de données d'entraînement. L'ajout d'un masque permet de ne rendre visible au modèle que les $(k-1)$ premiers embeddings de la *ground truth* afin de l'entraîner à prédire le $k^{\text{ème}}$ comme s'il les avait tous parfaitement prédits jusqu'au $(k-1)^{\text{ème}}$, et de cacher la suite de la séquence attendue. On note X' la matrice obtenue en sortie de ce mécanisme.
2. Le second mécanisme réalise une *multi-head attention* classique, mais en combinant en entrée X' avec la sortie Y de l'encodeur. L'équation de cette procédure est toujours (1), mais où

$$Q \triangleq W_Q X', \quad K \triangleq W_K Y \quad \text{et} \quad V \triangleq W_V Y \ .$$

Le reste de l'action du décodeur est décrit dans la figure.

Tailles des modèles Un Transformer repose sur l'empilement de plusieurs encodeurs et plusieurs décodeurs, possédant chacun un ou deux mécanismes de multi-head attention dont les matrices peuvent être de très grande taille. Le nombre total de poids à entraîner dans un modèle comme celui-ci est donc très important. GPT-3 possède 175×10^9 paramètres, et on estime que GPT-4 en possède environ 1.76×10^{12}.

III. *Instruction-tuning* : principe général et technique LoRA

III.1. Principe de transfer learning

Une démarche courante dans l'apprentissage en *machine learning* est le *transfer learning*. Il s'agit de découper l'entraînement d'un modèle en deux phases :

1. le *pretraining* est la ≪ pose des fondations ≫ du modèle. Un réseau de neurones est initialisé de manière aléatoire puis entraîné sur un large jeu de données afin d'acquérir une compréhension globale d'un

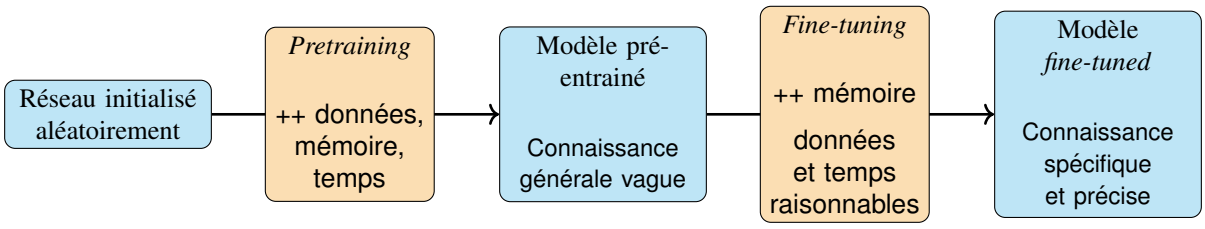

FIGURE III.1 – Principe du *transfer learning*.

phénomène. Pour les LLMs, un modèle est par exemple entraîné à acquérir une compréhension statistique d'une langue en particulier : le vocabulaire, la structure des phrases, les associations au sein de certains champs lexicaux, etc. Cette phase nécessite des moyens de stockage et de calcul importants.

2. le *fine-tuning* est la « touche finale » à apporter au modèle. Il se fonde sur les connaissances générales acquises lors du *pretraining* et termine l'entraînement du modèle sur un jeu de données bien plus petit. Ce jeu de données est généralement choisi pour concentrer le modèle sur une tâche particulière : analyse de sentiments, résumé de texte, conversation avec l'utilisateur... Cette phase nécessite des moyens de calcul moins importants (l'entraînement est moins long) mais le gain en mémoire n'est pas significatif dans le cas où le modèle est de grande taille, puisqu'il faut tout de même en charger tous les poids en mémoire.

Certaines techniques cherchent à réaliser le fine-tuning d'une partie seulement d'un réseau de neurones correctement choisie, pour réduire la charge en mémoire. Ces techniques sont appelées *Parameter-Efficient Fine-Tuning* (PEFT).

III.2. INSTRUCTION-TUNING

On appelle *instruction-tuning* une forme de fine-tuning qui spécialise les modèles de langage dans la compréhension d'instructions exprimées en langage naturel à la manière d'un CHATGPT. Ces instructions peuvent être diverses, aussi cette tâche reste assez vague.

III.3. MODÈLES LLAMA

Les modèles LLAMA [Tou+23] sont des Transformers. Leur structure est loin de ce qui est décrit dans la figure II.1, mais ils en conservent tout de même l'idée globale, notamment celle du mécanisme d'attention. Ces modèles ont été pré-entraînés par Meta AI et publiés en open-source en quatre versions : 7, 13, 30 et 60 milliards de paramètres.

III.4. LOW-RANK ADAPTATION (LORA)

Prérequis

— L'entraînement d'un réseau de neurones est un problème d'optimisation (descente de gradient, backpropagation).

— Décomposition en valeurs singulières d'une matrice.

Il a été observé que pour certains réseaux de neurones, lorsqu'ils sont entraînés sur des tâches spécifiques, l'entraînement d'une très faible portion de leurs poids résulte tout de même en de bonnes performances. Le problème d'optimisation des poids du réseau (descente de gradient, backpropagation) peut donc être résolu dans un sous-espace de dimension inférieure au nombre total de poids. La littérature dit que l'entraînement pour ces tâches est de *faible dimension intrinsèque*.

La technique de PEFT appelée *Low-Rank Adaptation* (LoRA) [Hu+21] repose sur ce principe. Considérons une matrice de poids quelconque d'un réseau de neurones avant entraînement, notée $W_0 \in \mathbb{R}^{k\times d}$. L'entraînement étant une simple modification de ces poids, on peut représenter cette matrice de poids après entraînement par $W_0 + \Delta W$ où $\Delta W \in \mathbb{R}^{k\times d}$. Puisque W_0 serait issue de la résolution d'un problème de faible dimension intrinsèque, l'intuition dicte qu'elle serait de *faible rang intrinsèque*. W_0 pourrait très bien être de plein rang numériquement, mais une partie de l'information qu'elle porte serait tout de même négligeable. On peut donc écrire la quasi-équation fondamentale de LoRA :

$$\Delta W \simeq BA \quad \text{avec} \quad B \in \mathbb{R}^{k\times r}, A \in \mathbb{R}^{r\times d} \quad \text{et} \quad r \ll \min(k,d) \ . \tag{2}$$

Le choix de r est laissé à la discrétion de l'utilisateur : c'est le premier hyperparamètre de LoRA. Cette équation prend plus de sens lorsqu'on la voit comme une décomposition en valeurs singulières tronquée. Posons $U \triangleq [u_1, ..., u_k] \in \mathbb{R}^{k\times k}, V \triangleq [v_1, ..., v_d] \in \mathbb{R}^{d\times d}$ et $\Sigma \in \mathbb{R}^{k\times d}$ les matrices canoniques de la décomposition en valeurs singulières de ΔW, i.e. $\Delta W = U\Sigma V^\top$. Supposons que les valeurs singulières σ_i soient rangées par ordre décroissant dans Σ. Alors on a :

$$BA = \sum_{i=1}^{r} \sigma_i u_i v_i^\top \ . \tag{3}$$

Attention, r n'est pas le rang de W_0 ! La décomposition en valeurs singulières entière de W_0 s'écrirait $\sum_{i=1}^{\mathrm{rang}(W_0)} \sigma_i u_i v_i^\top$ mais a priori $r \ll \mathrm{rang}(W_0)$. (3) est donc bien une décomposition tronquée. Rigoureusement, r n'est rien d'autre que l'indice de troncature de la décomposition complète, et donc $\mathrm{rang}(BA) \leq r$. Pourtant, les auteurs l'appellent le rang de la décomposition $\Delta W \simeq BA$. Dans les faits, ils choisissent pour r des valeurs très faibles $(4, 16, 32, 64)$, donc il est très peu probable que $\mathrm{rang}(BA) < r$. On peut finalement résumer LoRA :

$$\forall x \in \mathbb{R}^d, \quad W_0 x + \Delta W x \simeq W_0 + \frac{\alpha}{r} BAx \tag{4}$$

où :

— x représente une entrée quelconque d'une couche d'un réseau de neurones dont la matrice de poids avant entraînement est W_0 ;

— $\alpha \in \mathbb{N}$ est un coefficient qui constitue le deuxième hyperparamètre de LoRA.

Ainsi, lorsqu'un réseau subit un fine-tuning avec LoRA, ce sont les coefficients de B et A qui sont ajustés par descente de gradient et backpropagation. Ces coefficients sont en nombre bien inférieur à ceux de ΔW puisque r doit être choisi tel que $r \ll \min(k, d)$. Avant fine-tuning, B est initialisée comme la matrice nulle et A selon une loi normale centrée et de variance quelconque.

Application à un Transformer Pour réaliser le fine-tuning d'un Transformer, LoRA ne s'intéresse qu'aux matrices W_Q, W_K, W_V et W_O de chaque couche d'attention du réseau. Chacune des matrices est remplacée par une décomposition de forme (4).

IV. Évaluation des modèles de langage

Il existe plusieurs tests automatisés pour évaluer les performances d'un modèle de langage. Nous avons utilisé les tests suivants :

Massive Multitask Language Understanding (MMLU) est un ensemble de questions à choix multiples réparties en 57 catégories, allant de domaines élémentaires à hautement spécialisés (mathématiques élémentaires, histoire des États-Unis, droit, management, algèbre abstraite, philosophie, etc.) ;

BIG-Bench Hard (BBH) est un ensemble de tâches de logique mêlant des questions à choix multiples et des *Vrai ou Faux*. Ce jeu de données se concentre sur la reproduction d'un cheminement de pensée dans la façon dont les questions sont posées. Il permet de tester la capacité du modèle à effectuer une forme de raisonnement ;

Discrete Reasoning Over Paragraphs (DROP) est un jeu de 96 000 questions évaluant la capacité d'un modèle à lire de longs paragraphes et à raisonner à partir de ceux-ci. Les questions portent sur des éléments spécifiques des paragraphes et nécessitent d'en saisir la sémantique, parfois même de réaliser des calculs pour parvenir à une réponse numérique ;

HumanEval évalue la capacité d'un modèle à générer du code dans le langage Python. Il groupe 164 requêtes sous forme de *docstrings* demandant au modèle d'écrire une fonction exécutant une tâche précise. Le jeu de données inclut du code Python appelant la fonction écrite par le modèle avec différentes entrées pour tester l'adéquation du code qu'il a proposé.

Pour tester les modèles, nous utilisons l'implémentation faite dans le dépôt Git du projet `InstructEval` [3].

3. https ://github.com/declare-lab/instruct-eval

```
 Below is an instruction
that describes a task,
paired with an input that
provides further context.
Write a response that
appropriately completes the
request.

### Instruction:
<instruction>

#### Context:
<context>

### Response:
```

(a) Entrée avec le champ `contexte`.

```
 Below is an instruction
that describes a task.
Write a response that
appropriately completes the
request.

### Instruction:
<instruction>

### Response:
```

(b) Entrée sans le champ `contexte`.

FIGURE V.1 – Formattage des entrées dans le script du projet *Alpaca*.

V. Démarche expérimentale

Nous effectuons l'instruction-tuning de LLAMA 2 (7 milliards de paramètres) avec la technique LoRA sur un jeu de données d'instructions (décrit plus loin). Notre objectif est d'optimiser 4 hyperparamètres de l'instruction-tuning avec `NOMAD` pour obtenir de bons scores sur les tests mentionnés ci-dessus. Les hyperparamètres optimisés sont :

— r et α dans LoRA ;

— la probabilité de *dropout* de l'optimiseur AdamW utilisé dans l'entraînement ;

— le *learning rate* de l'entraînement (pas de la descente de gradient dans l'optimisation des poids du réseau).

V.1. DONNÉES

Le détail des données utilisées est fourni dans l'article [Tri+23]. Chaque entrée est un triplet (`instruction`, `contexte`, `réponse`). Le champ `contexte` est optionnel et sert lorsque l'instruction porte sur une donnée additionnelle (e.g. si l'instruction consiste à résumer un texte, ledit texte est alors passé dans le champ `contexte`). Lorsque ces données sont utilisées, elles sont formattées par les scripts d'Alpaca et présentées au modèle sous la forme illustrée dans la figure V.1.

V.2. BOÎTE NOIRE

Notre boîte noire est séparée en deux phases :

1. instruction-tuning de LLAMA 2 avec LoRA et les hyperparamètres choisis par `NOMAD` ;
2. le modèle qui vient d'être entraîné est évalué par une procédure. Son score à l'évaluation est la sortie de la boîte noire et sert de fonction objectif.

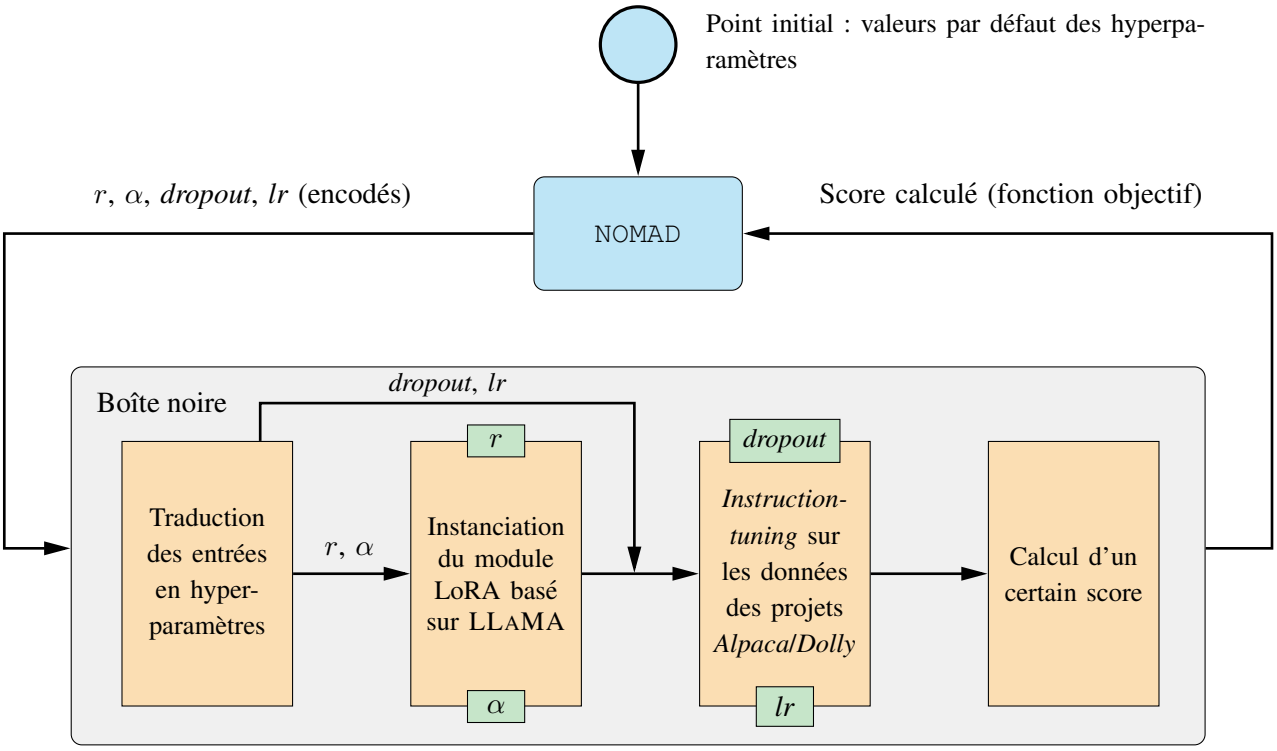

FIGURE V.2 – *Pipeline* correspondant à l'expérience #1. Les blocs verts marquent les endroits où les hyperparamètres sont utilisés. *lr* désigne le *learning rate*.

La façon dont la boîte noire est codée est détaillée dans le `README` de notre dépôt Git. La démarche complète est résumée dans la figure V.2. Un point choisi par `NOMAD` est noté comme un vecteur $\theta = (r, \alpha, dropout, lr)^\top$. Le point initial est choisi comme les hyperparamètres par défaut suggérés dans le cours de la plateforme HuggingFace [4] : $\theta^0 = (8, 32, 10^{-1}, 10^{-5})^\top$. Attention, pour de futures expériences, il faut considérer les HPs par défaut de la librairie PEFT : $\theta^0 = (8, 8, 0, 10^{-5})^\top$.

Le choix du score calculé pour servir de fonction objectif dépend des expériences.

VI. Résultats

Le détail des fonctions objectifs et des ensembles de valeurs possibles pour chaque hyperparamètre est donné dans le fichier `README` du dépôt Git et dans l'article. Ce que l'on appelle *loss* de validation d'un modèle (fonction objectif des expériences 2 et 3) est une mesure difficile à expliquer. Il s'agit d'une entropie croisée calculée sur les distributions de probabilité des tokens.

Dans la première expérience, nous avons utilisé le score MMLU de chaque modèle comme fonction objectif. Le souci est que cette démarche est difficilement soutenable telle qu'elle dans un contexte de machine learning. L'optimisation des hyperparamètres est une phase appelée *validation*, qui doit utiliser des données différentes des données d'entraînement et de test. Ainsi, si on utilise MMLU comme fonction objectif, on

4. https://huggingface.co/docs/peft/quicktour

ne peut plus utiliser ce score comme mesure de qualité dans une publication. On a optimisé le modèle pour maximiser ce score, c'est un biais inaceptable pour une publication en machine learning.

Les résultats détaillés et quelques directions de recherche possibles sont mentionnés dans le `README` du dépôt Git.

Références

[Hu+21] E.J. HU et al. *LoRA : Low-Rank Adaptation of Large Language Models*. 2021. DOI : 10.48550/arXiv.2106.09685. URL : https://doi.org/10.48550/arXiv.2106.09685.

[Tou+23] H. TOUVRON et al. *Llama 2 : Open Foundation and Fine-Tuned Chat Models*. https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/. Version v2. Meta AI, 2023. DOI : 10.48550/arXiv.2307.09288. URL : https://doi.org/10.48550/arXiv.2307.09288.

[Tri+23] C. TRIBES et al. *Hyperparameter Optimization for Large Language Model Instruction-Tuning*. 2023. DOI : 10.48550/arXiv.2312.00949. URL : https://doi.org/10.48550/arXiv.2312.00949.

[Vas+17] A. VASWANI et al. « Attention is All You Need ». In : *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, U.S.A. : Curran Associates Inc., 2017, p. 6000-6010. DOI : 10.48550/arXiv.1706.03762. URL : https://doi.org/10.48550/arXiv.1706.03762.