Introduction pratique aux modèles de language

Philippe Helluy

IRMA, Université de Strasbourg, Inria Tonus

August 28, 2025

Plan

Comment ça marche (en gros...)

Fine tuning of Qwen LLM for time series annotation

Comment ça marche (en gros...)

Très bref historique

- Les réseaux de neurones artificiels existent depuis **longtemps** ([?], fin des années 50)
- Des hauts et des bas, puis **Yann LeCun** ([?], reconnaissance d'écriture 1989)
- *Attention is all you need* [?], invention des **transformeurs** chez Google
- Sans une énorme **puissance** de calcul, ça ne marcherait pas.

Completion is all you need

- Principe: on se donne un début de texte. Il faut prédire le mot suivant. Exemple: "le chat mange le ..." (il faut deviner "mulot").
- Mots (ou *tokens*):

t_1	t_2	t_3	t_4	t_5	$t_6 = t_m$
.	chat	le	mange	matou	mulot

- Corpus: "le chat mange le mulot.", "le matou mange le mulot.", "..le chat mange.", "..le mulot mange.", "..le matou mange.", *etc.*
- Remarque: toutes les phrases ont $\ell = 6$ mots (on complète avec le mot bouche-trou ".").

Numérisation

- Codage: à chaque mot (ou *token*) on associe un vecteur à $m = 6$ dimensions

$$\text{"."} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \text{"chat"} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \text{"le"} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \textit{etc.}$$

- Plongement (*embedding*) dans un espace de dimension p plus petite (pour tenir compte des synonymes, entre autres). Par exemple $p = 5$. Le plongement E_{w_0} est une fonction de $\mathbb{R}^m$ à valeurs dans $\mathbb{R}^p$.
- Chaque token t_i est représenté par un vecteur v_i. Le vecteur w_0 des paramètres du plongement est inconnu.

$$v_i = E_{w_0}(t_i).$$

Encodeur

- Une phrase est donc représentée par un "tenseur" de ℓ vecteurs numériques collés les uns derrière les autres:

$$r_0 = v_{i_1} v_{i_2} \dots v_{i_\ell}$$

C'est donc un objet dans un espace à $N = \ell \times p = 30$ dimensions.

- La phrase passe dans k couches de transformeurs T_{w_i}, qui sont des applications de $\mathbb{R}^N$ à valeurs dans $\mathbb{R}^N$ avec des vecteurs de paramètres inconnus w_i

$$r_i = T_{w_i}(r_{i-1}), \quad i = 1 \dots k.$$

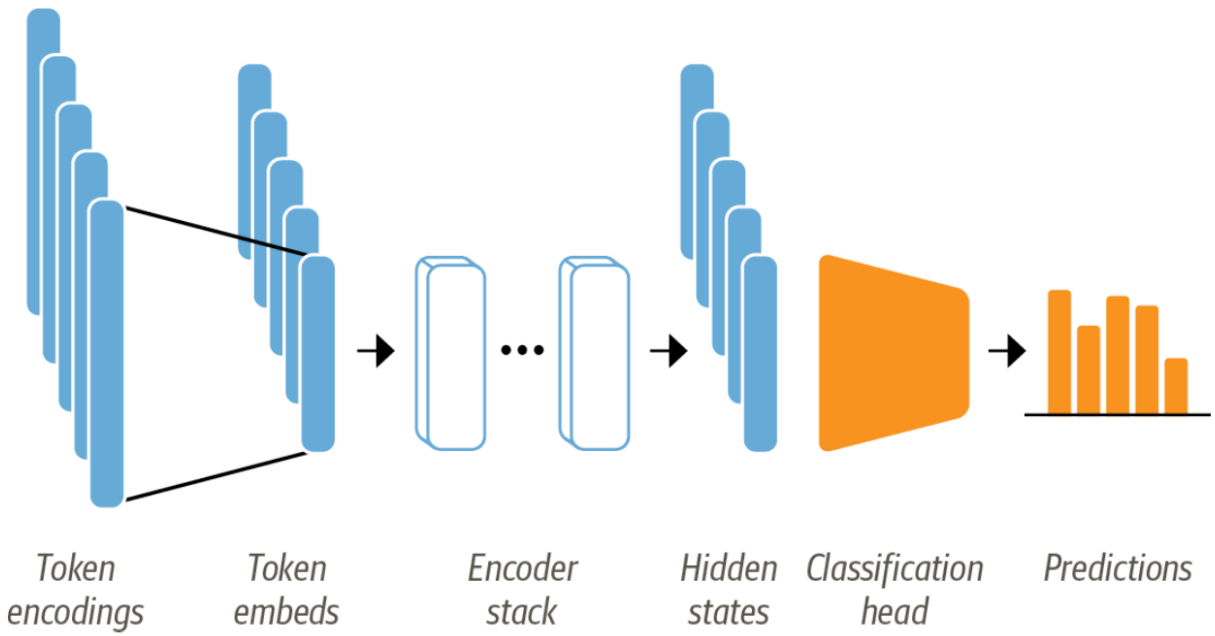
- Plus on s'enfonce dans les couches, plus la représentation r_i de la phrase initiale devient mystérieuse et abstraite. Le vecteur r_k contient l'information extraite par le réseau sur la phrase initiale r_0.

Décodeur

Enfin, le décodeur va permettre de prédire un vecteur de probabilités p dans $\mathbb{R}^m$: p_i est la probabilité que le mot suivant soit m_i.

$$p = D_{w_{k+1}}(r_k).$$

En résumé (plus de précisions dans [?, ?]):



Entrainement

- Choix de la forme des fonctions E_{w_0}, T_{w_i}, $D_{w_{k+1}}$: c'est un compromis entre coût, efficacité, simplicité. C'est un art autant que de la science pour l'instant.
- Historiquement plusieurs formes possibles: CNN, RNN, LSTM, transformeurs. Évolution fortement liée à la puissance de calcul disponible.
- Le vecteur des paramètres w, de taille s est **inconnu**.
- L'entraînement consiste à optimiser le choix de ces paramètres pour que le modèle retrouve au mieux les phrases du corpus.
- C'est la partie la plus difficile du calcul, il faut un super-calculateur, des processeurs spécialement conçus, ça coûte des millions d'euros.
- Ordres de grandeurs pour GPT-3: $\ell = 2000$, $m = 50000$, $p = 20000$, $s = 170$ milliard...

Inférence

- Une fois que les paramètres w sont calculés, l'inférence est rapide.
- On peut ré-entrainer un réseau pour une tâche spécifique, à coût réduit (*fine tuning*). Par exemple, Copilot et ChatGPT sont des versions spécialisées de GPT-3.
- Le *pre-prompt* est essentiel pour obtenir des résultats de qualité.
- Pour des raisons de coût de calcul, **ChatGPT n'apprend pas en temps réel**. Sa mémoire à court terme est donc limitée à quelques milliers de mots.

Exemple avec PyTorch

- PyTorch est une bibliothèque logicielle libre développée par la société Huggingface pour manipuler des réseaux de neurones.
- Exemple minimal d'utilisation.

Fine tuning of Qwen LLM for time series annotation

Objectives

- Teach a small LLM (Qwen2.5 0.5 billions parameters) to describe a time series
- Example of prompt:

```
Describe the time series in three sentences.
First sentence: describe trend (increasing/decreasing/flat).
Second sentence: noise intensity (low/medium/high).
Third sentence: approximate localisation of global maximum (beginning/middle/end)
and global minimum (beginning/middle/end).
Put the description in a JSON format with the following pattern
<json>{ "trend": <sentence1>,
"noise": <sentence2>,
"extrema": <sentence3> }
</json>
Series: [02, 01, 00, 03, ... ]
```

- Tokenization is essential: the values in the series are scaled to the range 00..99.

Necessity of training

- Small LLM fails to answer properly. It generates total garbage.
- Example of output:

```
series name: \"series\"
time interval: 1 day
``` To provide a detailed analysis of the given time series data, I will break it down into its com
### Time Series Description:
The provided time series consists of daily values for several variables over a period of one year (
### Trend Analysis:
- **Increase**: The values increase steadily from 01 to 04, then decrease again.
- **Decrease**: The values decrease from 04 onwards
...
```

- Let's try to improve this with supervised fine tuning
```

Practical methodology

- Generate a dataset of correct examples with a large LLM (Mistral, ChatGPT, etc.)
- Apply a supervised fine tuning (SFT) procedure on a small LLM from this dataset.
- In order to reduce the cost we adopt a LoRA approach. [1]
- Now go to `https://github.com/phelluy/DLAA_2025` and follow the README file.

[1] The LoRA (Low-Rank Adaptation) approach in supervised fine-tuning (SFT) freezes the original model weights and injects small trainable low-rank matrices into certain layers (typically linear projections in attention/FFN). This drastically reduces the number of parameters that need updating, making fine-tuning large models much more memory- and compute-efficient while still achieving strong adaptation.