Reinforcement Learning from Human Feedback and ChatGPT

Riccardo Brioschi

25 March 2024

Foundations of Reinforcement Learning

Language Modelling

- **Language Modelling** is the task of predicting what word comes next;
- More formally, given a sequence of words $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$,, compute the probability distribution of $x^{(t+1)}$:

$$P(x^{(t+1)} | x^{(t)}, \ldots, x^{(1)})$$

The Old Approach: Statistical NLP

In order to learn a Language Model, we can learn a **n-gram Language Model**.

- **Definition**: An n-gram is a chunk of n consecutive words;
- **Idea**: Collect stats about how frequent different n-grams are and use these to predict the next word.

  - unigrams: “the”, “students”, “opened”, ”their”
  - bigrams: “the students”, “students opened”, “opened their”
  - trigrams: “the students opened”, “students opened their”
  - four-grams: “the students opened their”

Figure: Examples of n-grams

The Idea in Practice

We start by making a **Markov assumption**: $x^{(t+1)}$ only depends on the preceding (n-1) words (to complete the n-gram). Therefore, we have:

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$

(assumption)

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}$$

(definition of conditional prob)

And we estimate this quantity by computing

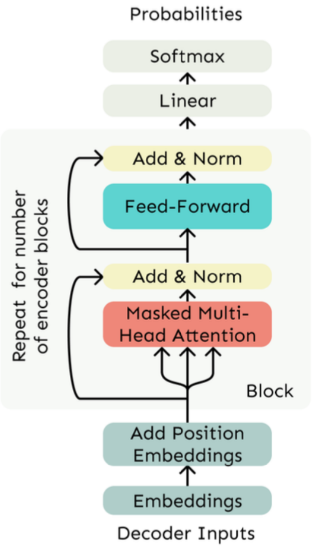
$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

(statistical approximation)

The "New" Approach: Transformers

The Transformer Decoder Architecture:

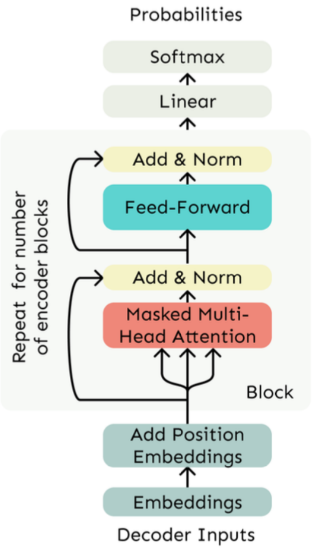
- Stack of Trasformer Decoder Blocks;
- Each block consists of:
  - (Masked) Self-Attention
  - Add & Layer Normalization
  - Feed-Forward Layer
  - Add & Layer Normalization
- Given an input sentence, the decoder generates the output using **Auto Regressive Generation**.



Step I: Tokenization

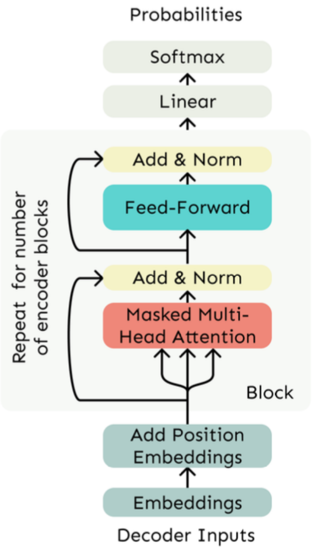
Idea: Use a Tokenizer and a huge matrix to obtain embeddings for each textual unit (tokens).

- Define a vocabulary V;
- Divide the input sentence into fundamental units (**tokens**) in V;
- Convert each token into a vector (**Embedding**);
- While training, learn the vector representation corresponding to each token (column of the matrix);
- Special tokens are usually added.



Step II: Position Embedding

- The order of tokens and words in a sentence is important;
- Fix the maximum input sequence length that the model can handle;
- Define a learnable vector for each position (**Position Embedding**);
- Perform element-wise addition between position embeddings and word embeddings;



The Goal

- **Remember:** We want to define a model capable of generating a textual output which is related to the given input sentence;
- In order to generate human-like language, it is fundamental to find a way to relate each token (word) to the others;
- More precisely, as we would like to predict the future tokens, we need a way to relate each token to itself and the preceding ones.
- This is obtained performing **Masked Attention**

Step III: (Masked) Attention Mechanism

- Let $\mathbf{w_{1:n}}$ be a sequence of input words in V;
- For each $\mathbf{w_i}$, define $\mathbf{x_i} = \mathbf{Ew_i} + \mathbf{p_i}, \mathbf{x_i} \in \mathbf{R^d}$ the word vector representation obtained combining position and word embeddings;
- Transform each embedding into query, key and value representations with weight matrices Q,K,V $\in R^{d \times d}$;

$$\boldsymbol{q}_i = Q\boldsymbol{x_i} \text{ (queries)} \qquad \boldsymbol{k}_i = K\boldsymbol{x_i} \text{ (keys)} \qquad \boldsymbol{v}_i = V\boldsymbol{x_i} \text{ (values)}$$

- Compute pairwise similarity between queries and keys; normalize with softmax.

$$\boldsymbol{e}_{ij} = \boldsymbol{q}_i^\top \boldsymbol{k_j} \qquad \boldsymbol{\alpha}_{ij} = \frac{\exp(\boldsymbol{e}_{ij})}{\sum_{j'} \exp(\boldsymbol{e}_{ij'})}$$

- Compute output for each word as a weighted sum of values.

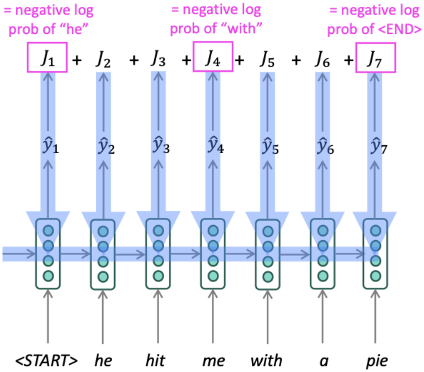
$$\boldsymbol{o}_i = \sum_j \boldsymbol{\alpha}_{ij} \boldsymbol{v}_i$$

Attention Mechanism - Recap

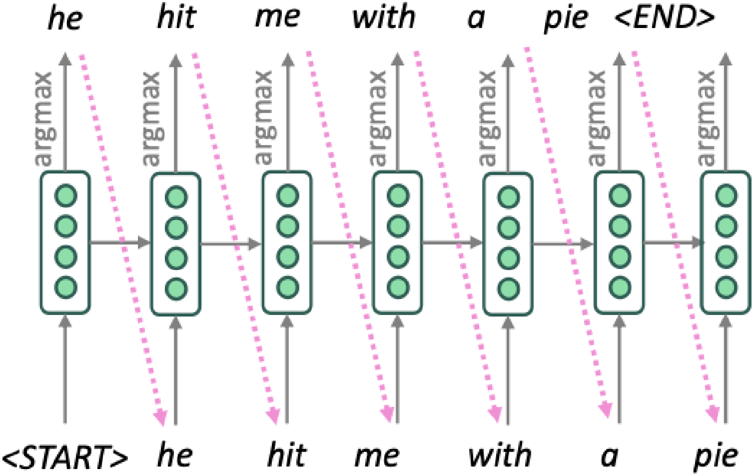
- For each input vector representation, we find a new embedding;
- This new embedding contains information about how a token relates to other tokens;
- Without changing the number of vectors, we increase the quality of the information they carry;
- This operation, together with Layer Normalization, Skip Connections and MLP is repeated multiple times;
- **Problem:** this procedure scales quadratically with the number of tokens.

Step IV: Training

- After the whole computation, we obtain vectors $\in R^{|V|}$;
- Softmax is applied, to get a proabability distribution;
- We define a loss term for each token;
- The loss term corresponds to the usual logistic loss.



Step V: Deploying the Model



Final Summary on Transformers

- Tons of documents and sentences are needed to train these architectures;
- Usually, **a model is pre-trained once, and then it is fine tuned depending on the task**;
- There are several methods to fine tune a model in efficient ways (PEFT, Prompt Tuning);
- Transformers are so famous because **they can handle images and audio** with the same mechanism.

RLHF: An Overview

- Despite the loss and lots of metrics, it is hard to ensure the generated output is a "good" text;
- It would be great to use **human feedback not only as a measure of performance but as a loss to optimize!**
- This is the idea behind RLHF: Reinforcement Learning from Human Feedback.

RLHF: An Example

LE

Can you explain to me how RL with human feedback works for language models like I am five?

Sure! So, imagine you have a robot named Rufus who wants to learn how to talk like a human. Rufus has a language model that helps him understand words and sentences.

First, Rufus will say something using his language model. For example, he might say "I am a robot."

Then, a human will listen to what Rufus said and give him feedback on whether it sounded like a natural sentence a human would say. The human might say, "That's not quite right, Rufus. Humans don't usually say 'I am a robot.' They might say 'I'm a robot' or 'I am a machine.'"

Rufus will take this feedback and use it to update his language model. He will try to say the sentence again, using the new information he received from the human. This time, he might say "I'm a robot."

RLHF: The Ingredients

In order to make this magic happen, we need to :

1. Pre-Train a **Language Model** π^{SFT};
2. Train a **Reward Model** on gathered data;
3. **Fine-Tune** the Language Model with RL, using the Reward Model to generate reward.

Training a Reward Model

The reward model should reflect human preferences, assigning scalar rewards to each generated output.

- Using the pre-trained model, generate pairs of answers $y_1, y_2 \sim \pi^{SFT}(x)$ given an input x;
- Let humans annotate pairs of answers, therefore defining a rank of preferences;
- Assume a latent structure for the Reward Model, such as

$$p(y_2 \preceq y_1) = \frac{exp(r(x, y_1))}{exp(r(x, y_1)) + exp(r(x, y_2))}$$

- Parametrize r and train the reward model, using the ranking from the users to define the usual logistic loss.

Training a Reward Model: Recap

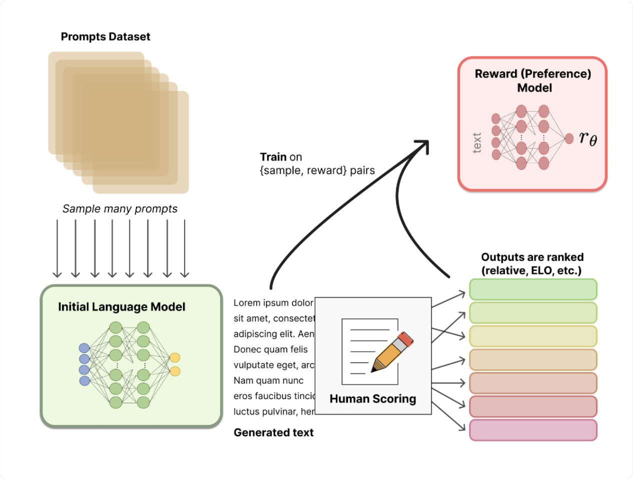


Figure: Training a Reward Model

Fine-Tuning using RL (I / II)

Before starting, it is necessary to formulate the fine-tuning task as a RL problem.

- **Policy**: the Language Model (it is indeed a probability distribution as we noticed);
- **Action Space**: the set of tokens that can be generated given the input x;
- **Reward**: a combination of the reward model and constraints on policy shift.

Fine-Tuning using RL (II / II)

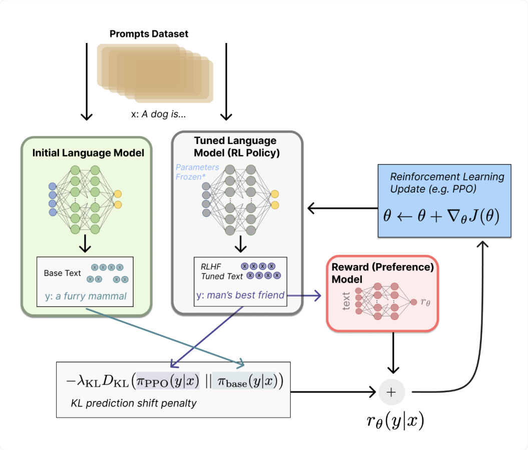


Figure: RLHF Schema

Bibliography

- Stanford CS224n: NLP with Deep Learning;
- "Attention is all you need", Vaswani et al.
- "Improving Language Understanding by Generative Pre-Training", Radford et al.
- "Direct Preference Optimization: Your Language Model is Secretly a Reward Model", Rafailov et al.
- Illustrating Reinforcement Learning from Human Feedback (RLHF), Hugging Face