Large language models

Herman Kamper

GPT is just a transformer language model

From language model to assistant

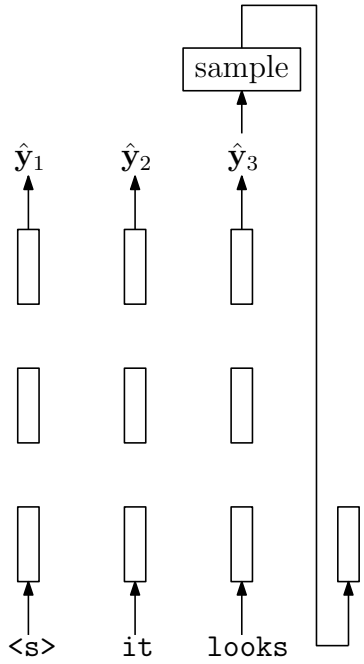
More on RLHF

High-level introduction

Intro to large language models by Andrej Karpathy [slides]

GPT is just a transformer language model

Inference: Predict one word at a time and feed it back in

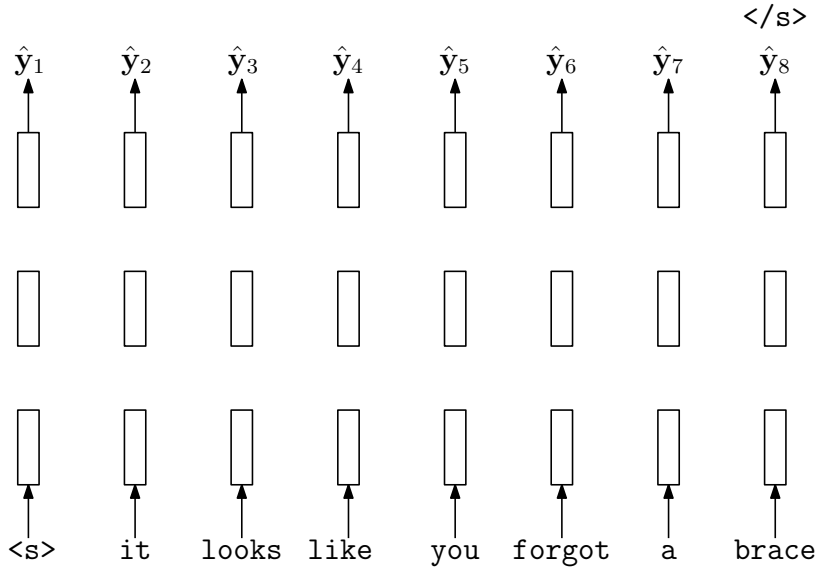


The model outputs the next word and then feeds that in as its own input to the next time step. In general, a model that takes its own output as input is called **autoregressive**.

There are different ways of sampling the output word from the predicted output distribution:

- Take the maximum
- Sample using the predicted probabilities
- Sample only from top-k highest probabilities

Training time: Predict all the next words and backprop



Masking ensures that the model cannot look into the future for predicting the output at the current time step. I.e. what happens during training resembles what happens during inference.[1]

This is not an encoder-decoder model: it's a decoder-only model.

My illustrations are with words, but the language model (LM) probably uses subword units like BPE. The basic architecture, I think, is still the one from (Radford et al., 2018).

Why is GPT so much better than previous LMs?

- Size of model
- Size of data: Compression of the internet (Karpathy)

[1] See the explanation of masking in my transformers note.

From language model to assistant

Three steps to go from GPT to ChatGPT:

1. Pretraining: Next-word prediction as above (GPT)
2. Finetuning on expert examples
3. Reinforcement learning from human feedback (RLHF)

2. Finetuning on expert examples

Get high-quality data and add <prompt> and <answer> tags:

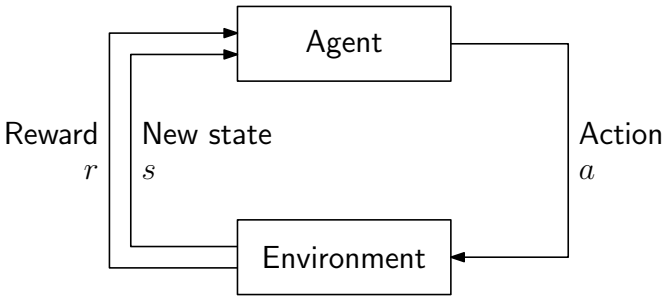
```
<prompt>
Can you help me with this code? It seems like there is
a bug.
print("hello world)
</prompt>

<answer>
It looks like you forgot to close the string passed to
the function print. You have to add a closing quote to
properly terminate the string. Here is the corrected
function:

print("hello world")

Let me know if I can help with anything else!
</answer>
```

You can think of this as supervised learning, but really it is still exactly the same task as the task used during pretraining. The dataset is just swapped out and training is continued.

It is a little bit amazing that all of the knowledge gained during pretraining isn't wiped out.

3. Reinforcement learning from human feedback (RLHF)

Reinforcement learning (RL):

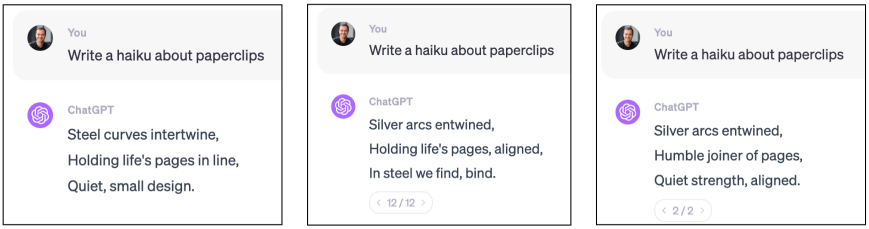


In our case the action is an answer $a = \mathbf{y}_{1:M}$ to a given prompt $s = \mathbf{x}_{1:T}$ and we want to update the language model so that it outputs high-reward answers (i.e. the type of answers a human would give).

But what reward function should we use?

We learn a reward function by asking humans to rank possible outputs:[2]

Simple example: it's much easier to spot a good haiku than it is to generate one:



The result is even more assistant-like behaviour.

But Rafailov et al. (2023) and others say that it isn't really necessary to use reinforcement learning to update the model: you can do it directly through supervised learning by backpropping through the learned reward model.

[2] Figure by Andrej Karpathy.

More on RLHF

Learning a reward model from humans

For prompt $\mathbf{x}_{1:T}$, the reward model $h_{\boldsymbol{\phi}}(\mathbf{x}_{1:T}, \mathbf{y}_{1:M}) \in \mathbb{R}$ should output

- Large positive values for good answers $\mathbf{y}_{1:M}$
- Large negative values for bad answers $\mathbf{y}_{1:M}$

To train the model, we use a dataset of prompts each with candidate answers. The answers can be generated from our current LM or we can ask human experts to write answers (or a mixture of these).

Say we have $B = 3$ candidate answers:[3]

$$\mathbf{y}^{(1)}_{1:M^{(1)}} \qquad \mathbf{y}^{(2)}_{1:M^{(2)}} \qquad \mathbf{y}^{(3)}_{1:M^{(3)}}$$

We then get a human to select their favorite answer. Let $b \in \{1, 2, \ldots, B\}$ indicate the selection.

We then train $h_{\boldsymbol{\phi}}$ by modelling the probability of the selection as

$$P_{\boldsymbol{\phi}}\left(\text{selected} = b \,\middle|\, \mathbf{x}_{1:T}, \{\mathbf{y}^{(k)}_{1:M}\}_{k=1}^{B}\right) = \frac{e^{h_{\boldsymbol{\phi}}(\mathbf{x}_{1:T}, \mathbf{y}^{(b)}_{1:M})}}{\sum_{j=1}^{B} e^{h_{\boldsymbol{\phi}}(\mathbf{x}_{1:T}, \mathbf{y}^{(j)}_{1:M})}}$$

I.e. $h_{\boldsymbol{\phi}}$ are the logits in a softmax classifier.

We train the classifier using the negative log likelihood loss. With one training example consisting of a prompt $\mathbf{x}_{1:T}$, B candidate answers $\{\mathbf{y}^{(k)}_{1:M}\}_{k=1}^{B}$ and the selection b, the loss is

$$\begin{aligned} J(\boldsymbol{\phi}) &= -\log P_{\boldsymbol{\phi}}\left(b \,\middle|\, \mathbf{x}_{1:T}, \{\mathbf{y}^{(k)}_{1:M}\}_{k=1}^{B}\right) \\ &= -\log \frac{e^{h_{\boldsymbol{\phi}}(\mathbf{x}_{1:T}, \mathbf{y}^{(b)}_{1:M})}}{\sum_{j=1}^{B} e^{h_{\boldsymbol{\phi}}(\mathbf{x}_{1:T}, \mathbf{y}^{(j)}_{1:M})}} \end{aligned}$$

[3] Each candidate answer will have a different length $M^{(j)}$ that I denote explicitly here but will now drop.

RL using the learned reward model

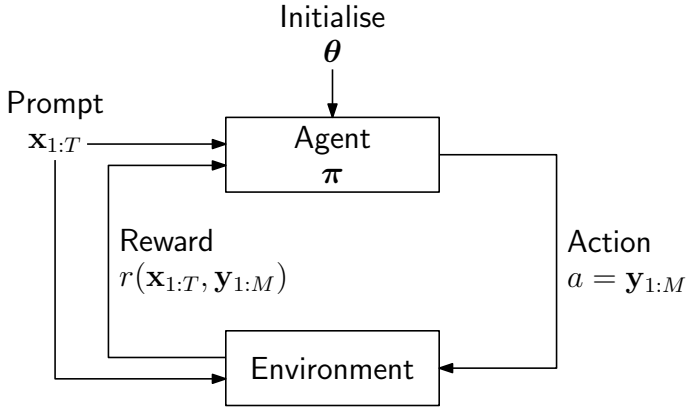
We denote the parameters of our LM before RLHF as $\boldsymbol{\theta}$.

Intialise the parameters of the RL agent LM: $\boldsymbol{\pi} \leftarrow \boldsymbol{\theta}$

As actual reward function we combine the learned reward model and a penalty so that the agent LM doesn't drift too far from the original:

$$r(\mathbf{x}_{1:T}, \mathbf{y}_{1:M}) = h_{\boldsymbol{\phi}}(\mathbf{x}_{1:T}, \mathbf{y}_{1:M}) - \beta \log \frac{P_{\boldsymbol{\pi}}(\mathbf{y}_{1:M}|\mathbf{x}_{1:T})}{P_{\boldsymbol{\theta}}(\mathbf{y}_{1:M}|\mathbf{x}_{1:T})}$$

Normally in RL we have cycles of taking an action, getting a reward and moving to a new state. Here we have just one episode where for a prompt $\mathbf{x}_{1:T}$ we generate an answer $\mathbf{y}_{1:M}$ and get reward r:



The agent LM parameters $\boldsymbol{\pi}$ are updated using proximal policy optimisation (PPO). I won't get into this here, but you can have a look at these resources to understand this RL learning approach:

- PPO theory by Ehsan Kamalinejad
- Proximal policy optimization explained by Edan Meyer

See (Ziegler et al., 2020) for further details on RLHF.

Why learn a reward model instead of just asking humans?

An alternative to learning a reward model beforehand would have just been to ask humans directly to rate outputs as our model produce it. We could then get the reward directly (and probably more accurately) in our RL loop. But this would be much much slower than first learning a reward model, fixing it, and then running the RL loop very quickly (since we don't have to wait for a slow human to respond).

Videos covered in this note

to-do

Further reading

I found Andrej Karpathy's blog post Deep reinforcement learning: Pong from pixels helpful.

References

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI*, 2018.

R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," in *NeurIPS*, 2023.

D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving, "Fine-tuning language models from human preferences," *arXiv*, 2020.