W. ROSS MORROW

1. Introduction

Consider a corpus $\mathcal{C}$ of tokens from a language $\mathcal{T} = \{1, \dots, T\}$ (w.l.o.g.) that at the very least "covers" $\mathcal{T}$ in the sense that every $t \in \mathcal{T}$ appears in $\mathcal{C}$. By a corpus we basically mean, here, a giant body of text or a "long" sequence of elements of $\mathcal{T}$. Can we compute the empirical distribution of next tokens for some batch size B? Let's define what we mean clearly: we want to estimate the conditional probabilities

$$\rho_B(\mathbf{t}, \tau) = \mathbb{P}(\ t_{B+1} = \tau \mid t_1, \dots, t_B\ )$$

with "(Conditional) Empiricial Frequencies" or (C)EFs. This is, of course, a purely Markov representation of token sequences.

Before detailing how, why would we do such a thing? Markov modeling of language aged out forever ago. Yet a plausible "hypothesis" is that these probabilities are what LLMs like GPT are fundamentally modeling. Any functional representation, such as a simple variant of multiclass logit regression or the deep networks in modern LLMs, impose structure on sequential predictions. Such structure exists to "smooth" or "fill" gaps in knowledge of some "true" conditional probabilities for "small" data volumes (relatively). As the training corpus size grows a model from a suitably "universal" class token sequence predictor estimated consistently will increasingly tend to the (C)EFs and effectively "regurgitate" patterns from its training data.[1] We mostly ignore the mathematics of the underlying statistical assumptions here (consistency, unbiasedness).

This idea – that the best a LLM could do is defined by the (C)EFs – has implications for conceptualizing what LLMs are. First, should this hypothesis be correct, it is a formal, almost philosophical illustration of how it is impossible for LLMs to be generatively "innovative". In a strict sense they can only repeat the patterns of the past, as described by the (C)EFs in the training data. Which is not to say they are not still *useful*, by any means; after all, who is to say we ourselves don't just resample the past. We should still be able to reflect on the structure and limitations of such tools, as well as ourselves. Second, if model structure aims at universality, and there is enough data to faithfully represent the (C)EFs, then the structure has to *compress* the (C)EFs. In slightly plainer words: as we increase the volume of data used to train a LLM from a reasonably "universal" class, either

[1] An empirical distribution (from iid samples) converges w.p.1 to the true distribution from which samples are drawn. We can't formally speak of "consistency" of a NN-LLM as its model and parameters are fictitious, but we can safely suppose the point is to generate a distribution with a similar target, perhaps meaning an empirical distribution of samples from the LLM has the same convergence property.

(a) estimating the LLM parameters must be vastly simpler than computing the (C)EFs directly or (b) sampling sequences from the LLM must be vastly simpler than from the (C)EFs (or both). Something like GPT-3 is hardly "parsimonious" being based on over 175 *billion* parameters requiring O(1TB) to describe, with yet larger models also publicized, and generative sampling requiring complex distributed computing on specialized hardware. This is not to say we actually *have* enough data to fully describe all sayable things (far from it, probably), just to point out an inevitability of the current trend towards more and more data.

At least to me, it is not entirely clear that (C)EFs are "efficiently" computable (in, so to speak, "training"), or sampling from them is efficient, or, most importantly, if samples from (C)EFs appear to mimic "real" language structure like LLMs do (as defined in the training corpus). Here we'll revisit history and focus on those questions. What datastructure models (C)EFs? How would we (efficiently, scalably) compute them from a (large) corpus? How would we generate sequences? Are those sequences at all representative of text in the training corpus?

2. Modeling

Again we want to estimate a Markov model of token sequences via the (C)EFs

$$\rho_B(\mathbf{t},\tau) = \mathbb{P}(\ t_{B+1} = \tau \mid t_1, \dots, t_B\ )$$

This is probably to be done via something like

$$\hat{\rho}_B(\mathbf{t},\tau) = \frac{\#\text{ of occurrences of }\mathbf{t}\circ\tau\text{ in }\mathcal{C}}{\#\text{ of occurrences of }\mathbf{t}\text{ in }\mathcal{C}}$$

where $\circ$ is concatenation. Why *something like*? The question would be whether *exactly* this empirical value would be *generative* without an underlying structural model. Simply, suppose we have some $\mathbf{t}$, we generate a next τ, but such that $(t_2, \dots, t_B, \tau)$ is not in $\mathcal{C}$. Strictly speaking, we can't then use literal B-prefix conditional empirical distributions as they would not tell us what comes next. However we'll stick to the naivest possible calculation, and outline a well-defined sampling process below.

Clearly $\rho_{B-1}(\mathbf{t},\tau)$ is a *marginal* relative to $\rho_B(\mathbf{t},\tau)$, because ρ_B is more specific than ρ_{B-1}. That is,

$$\rho_{B-1}(\mathbf{t},\tau) = \mathbb{P}(\ t_B = \tau \mid t_1, \dots, t_{B-1}\ ) = \sum_t \rho_0(t)\rho_B\big(t\circ\mathbf{t},\tau\big)$$

where $\rho_0(t) = \mathbb{P}(T = t)$. Also

$$\hat{\rho}_0(\tau) = \frac{\#\text{ of occurrences of }\tau\text{ in }\mathcal{C}}{|\mathcal{C}|}$$

which we are assured is defined and positive (because the corpus covers the token language).

So how would we compute $\hat{\rho}_B$? To sketch a start, we can do this in a single pass over $\mathcal{C}$ as follows:

(0) Initialize a hashmap $\mathcal{P}_B$ whose keys are B-token strings and values are also hashmaps with single-token keys and whose values are floats (technically `doubles`). We will store $\hat{\rho}_B(\mathbf{t},\tau) = \mathcal{P}_B[\mathbf{t}][\tau]$. Initialize $c = B$.
(1) Set $\mathbf{t} = \mathcal{C}[c - B : c)$, $\tau = \mathcal{C}[c]$
(2) If $\mathcal{P}_B[\mathbf{t}]$ does not exist, initialize $\mathcal{P}_B[\mathbf{t}]$ as needed, and set $\mathcal{P}_B[\mathbf{t}][\tau] = 1$. Otherwise, if $\mathcal{P}_B[\mathbf{t}][\tau]$ does not exist, set $\mathcal{P}_B[\mathbf{t}][\tau] = 1$. Otherwise increment $\mathcal{P}_B[\mathbf{t}][\tau]$.
(3) Increment c and go back to (1) unless $c = |\mathcal{C}|$, in which case continue to (4).
(4) For all keys $\mathbf{t}$ defined in $\mathcal{P}_B$, compute

$$S(\mathbf{t}) = \sum_{\tau \in \mathcal{P}_B[\mathbf{t}]} \mathcal{P}_B[\mathbf{t}][\tau]$$

and update

$$\mathcal{P}_B[\mathbf{t}][\tau] \leftarrow \mathcal{P}_B[\mathbf{t}][\tau] / S(\mathbf{t})$$

Technically this is a bit like a double-pass algorithm considering the normalization in (4), but it is still $O(|\mathcal{C}|)$. We also might want to reverse the ordering of tokens in the keys of $\mathcal{P}_B$, to enable easier search with some tools that can do bulk return with partial key matching, but that's a detail. This is also embarassingly parallelizable, distributing the right overlapping subsets of $\mathcal{C}$ and merging globally, though we don't outline the details.

Now, we also need to reduce to $P_{B-1}, P_{B-2}, \dots, P_0$ for any hope of prediction. Specifically, we could predict a token τ for which the concatenated subsequence $(\mathbf{t}[2:]) \circ \tau$ does not occur in the corpus. By assumption P_0 exists and is "complete", as the empirical frequencies of tokens in the corpus are defined by virtue of covering. That is, we can always simply sample from the simple occurrence likelihood. Our process could be to take find longest suffix of $\mathbf{t}$ that exists in the corpus,

$$\mathbf{t}[:-k] \quad \text{where} \quad k = \arg\min_{0 \le k \le |\mathbf{t}|} \{ |\mathbf{t}[:-k]| : \mathbf{t}[:-k] \subset \mathcal{C} \}$$

and sample from $P_{|\mathbf{t}[:-k]|}[\mathbf{t}[:-k]]$. In the "worst case" $k = |\mathbf{t}|$ and we choose from P_0.

Prediction actually means a $\mathcal{T}$-set of suffix trees may be more suitable, where we access $P_B[\mathbf{t}]$ via following the "reverse" or suffix path

$$t_{|\mathbf{t}|} \to t_{|\mathbf{t}|-1} \circ t_{|\mathbf{t}|} \to \cdots$$

from a (guaranteed-to-exist) root $t_{|\mathbf{t}|}$ to the deepest accessible node. This could also aid in the "marginalization" process. Each node would contain a hashmap representing the distribution over next most likely tokens.

Presuming we have the (C)EFs, how can we "validate"? We quote "validate" presuming that the (C)EFs are all there is, so when new sequences are observed they are more like new information than test samples to be predicted. (This data-centric view is wrong, of course. There is higher level structure in language, whose rendering in a model is likely hard to capture without rule-based models especially with naive character-based tokenizations. More advanced "language-aware" tokenizers might fare better.) In any case, we could view "validation" as a joint distribution prediction task: Specifically, given a novel sequence $\mathbf{t} \circ \tau$, what is the probability of predicting this sequence? If $\tau \in \mathcal{P}_B[\mathbf{t}]$, then the probability is estimated by $\mathcal{P}_B[\mathbf{t}][\tau]$; otherwise if $\tau \in \mathcal{P}_{B-1}[\mathbf{t}]$ it is $\tau \in \mathcal{P}_{B-1}[\mathbf{t}][\tau]$; and so on down to

$\mathcal{P}_0[\tau]$. The closer this is to 1, the "better" the (C)EF model is. But of course, "better" is a strange adjective here, as it means something more like how *un*-novel the new sequence actually is.

3. A Simple Example

The code 'efnlp' takes a perhaps shockingly naive approach to modeling (C)EFs. We use 'dict'-based suffix trees to parse a (character) tokenized corpus of text, and generate text with a best-matching-marginal sampling approach as discussed above.

Some researchers use the full corpus of Shakespeare's writing to quickly analyze language models. Here is the start of sample of 10k token long generated Shakespeare using (C)EFs with character tokens and 10-token sequences:

> Having the fearful'st time to chide.
>
> Nurse:
> Mistress, how mean you that? no mates for you,
> Unless you have lately told us;
> The Volsces
> May say 'This mercy we have spent our harvest of his coffers shall be joyful
> of thy company.
>
> ARCHBISHOP OF CARLISLE:
> Marry. God forbid! Where's Abhorson, there?
>
> ABHORSON:
> What, household Kates.
> Here comes a man

This is pitiful Shakespeare but not at all gibberish. For the most part the words are words, line starts are capitalized, there is punctuation, there are "character" headings before statement blocks suitable to a play, and "Abhorson" is even called for in one line and responds next. Honestly I find this output intriguingly realistic-ish for computing using only counts and ratios.

Here's the run command for that sample and (modified) outut:

Listing 1. Analyzing and generating some Shakespeare with (C)EFs

```
$ python -m efnlp -c data/tinywillspeare.txt \
        -m -b 10 -g 100000 -o sample-results.txt
[...:31:07.445610] Forming (character) language
[...:31:07.491561] Encoding corpus
[...:31:07.569177] Corpus is 1,115,393 tokens long
[...:31:07.569217] Parsing prefix/follower tokens
```

```
[...:31:41.985965] Normalizing to empirical frequencies
[...:31:51.631065] Memory (roughly) required: 62.4 MB
        (about 8,183,314 dbl, 16,366,628 fl)
[...:31:51.631112] Sampling and decoding 100000 tokens
[...:31:52.810630] Writing sampled results to \
        sample-results.txt
```

We shorten an ISO timestamp in the "logs" here, and first number in the log (`31`) is the minutes place. Note the entire exercise – encoding, estimating, and generating – completes in under a minute. Sampling 10k tokens takes about a second (1.2s, for about 0.1ms per token sampled). We more or less store $6x$ the corpus volume in (C)EF data, the equivalent of 8M texttttdoubles (16M `floats`). This is all without any attention to code optimization, in a single process, on a 4-year-old 2.3 GHz i9 with 16GB 267 MHz DDR4 RAM.

One take at a NN-LLM from a transformers example uses this data claims 10M parameters (so more memory if `doubles`, less if `floats`) and takes 15mins to estimate using a GPU (and SOTA software). Sampling time for generation is not available, nor is a sample.

4. Scaling

In an N-token long corpus $\mathcal{C}$ there are $N-B-1$ $(B+1)$-long subsequences of prefixes and (single token) successors. A simple estimate is that this requires, at most, storing $(N-B-1)(B+1+d)$ data elements where d counts data required for empirical frequencies (a counter, and/or a probability). Note this is not linear in $B \leq N-1$, and is maximized at $B_* = (N-2-d)/2$. This is an *overestimate* by missing two things: that only certain prefixes will exist in the corpus, and for those there will be a fixed set of successors. This *underestimates* any storage we require for capturing the marginals for matching based on less than B tokens in a prefix, although in principle those can computed on demand instead of cached (with slower sampling of course).

A more "realistic" (minimum) storage scaling concept is $(N-B-1)(B+1+d)r(B)s(B)$ where

$$r(B) = \frac{\#\text{ of prefixes}(N-1,B)}{N-B-1} \quad \text{and} \quad s(B) = \frac{\#\text{ of patterns}}{\#\text{ of prefixes}}$$

(leaving N implicit). Broadly speaking we (probably) know some features of these "functions" r and s. Specifically, $r(1) = L/N$ ($\ll 1$), and $r(B) \uparrow 1$; in fact $r(N-2) = 1$, because $\#$ of prefixes$(N-1, N-2) = 1$. Also $s(B) \geq 1$ (every prefix has at least one pattern) and $s(B) \downarrow 1$ (as longer prefixes are considered, their successors are more likely to be unique).

In the $1,115,393$ tokens Shakespeare we have the following parsing results:

B	prefixes	$(r(B))$	patterns	$(s(B))$	avg τ	min space	parse time	gen time
1	65	(0.0%)	1,403	(0.1%)	21.6	49kB	1s	0.3ms
2	1,403	(0.1%)	11,556	(1.0%)	8.2	264kB	2s	0.3ms
3	11,556	(1.0%)	50,712	(4.5%)	4.4	1MB	3s	0.4ms
4	50,712	(4.5%)	141,021	(12.6%)	2.8	4MB	5s	0.5ms
5	141,021	(12.6%)	283,313	(25.4%)	2.0	9MB	7s	0.6ms
7	447,352	(40.1%)	609,659	(54.7%)	1.4	26MB	16s	0.9ms
10	858,920	(77.0%)	937,254	(84.0%)	1.1	50MB	35s	1.0ms
12	991,391	(88.9%)	1,027,857	(92.2%)	1.0	63MB	51s	1.2ms
15	1,069,423	(95.9%)	1,081,060	(96.9%)	1.0	78MB	74s	3.0ms
20	1,103,358	(98.9%)	1,106,345	(99.2%)	1.0	101MB	144s	13.8ms

With 1-token prefixes (bigrams) the and quite a few successors per "prefix" (~ 22) stored in at least $49kB$; but the generative output from such a sparse model is, of course, unequivocally junk. The number of prefixes found increases but with diminishing returns; by 10- or 12-token prefixes we already have "most" (77% and 88.9% respectively) of the possible prefixes. The number of successors per prefix decreases (of course): with 5-token prefixes, we already have only 2 successors per prefix (on average) and by 10-tokens successors are on the whole unique. Note that this is a lower limit on how long our prefixes should probably be: by design every prefix will have at least one successor, so if we have (even on average) a single successor per prefix we have probably captured all there is to capture with (C)EFs.

Mountain View CA
Email address: `morrowwr@gmail.com`