Last Day: Core Mechanisms of Transformers & Assignment 3

Work-in-Flight: Assignment 3
- Project Approval-in-Principle.
- Project Proposal Document & Presentation

Today: 1. Language Generation using Transformers
2. Project Ideaation

Language Generation Using Transformers

Recall: A language model is trained to predict the next word after an input sequence of words.

So, if you can do that, then you can predict a whole sequence of words, one at a time, by taking each individual predicted word that is produced, appending it to the input sequence; generating the next word after that — and so on.

This is called "Auto-regressive" generation (is a general term used in many kinds of sequence predictors).

- S. Jurafsky, section 9.4.3, in context of RMUs.

E.g. Starts with: The clean river flowed.
Generate: into
Then next: The clean river flowed into.
Generate: the
Next: The clean river flowed into the
Generate: ...
Example Generation from GPT-2
See: https://huggingface.co/spaces/docs-demos/gpt2

1. From GPT-2 medium (mid-sized GPT-2 in terms of # parameters).

Input: The clean river flowed

Generation Parameters (Unknown! i.e. Temperature, Top-P, etc., max tokens generated)

Output: The clean river flowed in steady but strong currents.

After a week of hard-boiled fish hunt, all was tranquil again. They went out in a group on the small river and ate their dinner along the banks with some of the fish.

2. From GPT-2 XL (largest GPT-2).

Input: The clean river flowed

Generation Parameters (Unknown! i.e. Temperature, Top-P, etc., max tokens generated)

Output: The clean river flowed. We walked on to the other side with the people we left behind. We found a small restaurant with a bench near the river — a small oasis at the end of the world, really.

In the evening we
- see examples from GPT-2 on previous page

- recall the specific output of the language model, given an input sequence of embeddings:

\[ \begin{align*}
X_0, X_1, \ldots, X_{n-1} : \\
X_0 & \quad P(w_0) \\
X_1 & \quad P(w_1) \\
\vdots & \quad \vdots \\
X_{n-1} & \quad P(w_{n-1})
\end{align*} \]

where \( M \) is the size of the vocabulary.

- the probability that each word in vocabulary is the next word doesn't choose the word.

- for a given sequence input, which word is selected as output? Is called Decoding

1. "Greedy" method: select the highest probability word.
   - you'll see this in Assignment 3 part 2.
   - does not work well in general; obvious, but boring/un-interesting words are chosen: repetitive.
   - may choose the most likely next word but does not result in the most likely sequence of generated words.
     - gets stuck in a "local" optimum.

I.e. given input sequence \( X_0 \ldots X_{n-1} \) want the generated sequence \( Y_0 \ldots Y_{n-1} \) of \( k \) words to be most likely.
I.e. want \( P(Y_0) \times P(Y_1) \times \cdots \times P(Y_{n-1}) \) to be maximized.
but we don't know \( P(Y_i) \) when selecting \( Y_0 \) (or \( Y_j \) when selecting \( Y_j \)).

- this is a tough problem because there are \( M^k \) possible solutions.
  - e.g. \( M = 50,000 \), \( k = 20 \) is a big number of large NN inferences to make.
  - i.e. way too many.
- per Jurafsky, Section 10.5, think of the selection of the output sequence as a tree with probabilities at each layer.

Input: The clear river flowed...

Outputs, produced from many invocations of inference model:

Tree of Generation Probabilities

<table>
<thead>
<tr>
<th></th>
<th>W1</th>
<th></th>
<th>W2</th>
<th></th>
<th>W3</th>
<th>P(W1)P(W2)P(W3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>in</td>
<td></td>
<td>the</td>
<td></td>
<td>lake</td>
<td>.015</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>with</td>
<td></td>
<td>other</td>
<td>.006</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the</td>
<td></td>
<td>town</td>
<td>.012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>more</td>
<td></td>
<td>rivers</td>
<td>.004</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a</td>
<td>lake</td>
<td></td>
<td></td>
<td>.006</td>
</tr>
<tr>
<td></td>
<td></td>
<td>another</td>
<td></td>
<td>river</td>
<td>.001</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>there</td>
<td></td>
<td>was</td>
<td>.021</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>many</td>
<td></td>
<td>people</td>
<td>.006</td>
<td></td>
</tr>
</tbody>
</table>

- so the most probable W1 - "in" doesn't lead to the most probable sequence of W1 W2 W3.
- to get the optimal prob sequence is very hard.

2) Beam Search is a heuristic that prunes the full search tree a lot.
   - general descriptor: walk through the tree keeping the k-most probable sequences.
   - at each level, look at V possible new words for each of the k sequences.
- that produces $K \times V$ new sequences
- compute the probability for all by multiply probabilities
- keep the $K$ highest.

- go down to next level.
- each of the $K \times V$ a each level costs one inference run through model expensive!

- was thought to be best method, but isn't used it seems
- instead:

3. Nucleus Sampling:

- given the set of output probabilities of the next word — $P(w_0), P(w_1), \ldots, P(w_{n-1})$

- select the next word through a random process in which the probability of selecting word $w_i$ is $P(w_i)$ — how?

- how is that done? Show by example:

  suppose there are just 3 words with probabilities for the next word:

<table>
<thead>
<tr>
<th></th>
<th>$P(up)$</th>
<th>$P(down)$</th>
<th>$P(left)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>up</td>
<td>0.5</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>down</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>left</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  - if $R \leq 0.5$ choose up
  - $0.5 < R \leq 0.8$ choose down
  - $0.8 < R \leq 1.0$ choose left

  so up will be chosen with probability $0.5$, down with prob. 0.3
  $\frac{1}{3}$ left with prob. 0.2

\[ N = \text{vocabulary size} \]
- Simplification: only select from the top K most probable words (top-K sampling)

- Most widely used: top-\(P\) sampling: only select from those words that all together have the sum of probabilities = \(P\) (0 < \(P\) ≤ 1)
  \(P = 1\) means use all; often \(\alpha > 0.8\) vastly reduces the number considered

- Also a really unusual adjustment to the sampling method involves an adjustment to the softmax probability calculation at end of each inference.
  - recall \(\text{softmax}(\text{logits})\)
  - let \(L_i\) be the logits output of word \(i\).
  \[ P(w_i) = \frac{\exp\left(\frac{L_i}{\tau}\right)}{\sum_{i} \exp\left(\frac{L_i}{\tau}\right)} \]
  where \(\tau\) is a parameter call Temperature \(0 \leq \tau\)

- \(\tau = 1\) is normal sampling; \(\tau\) close to zero is greedy
- \(\tau < 1\) makes the higher probability words more likely (because exponential works against smaller values).
- \(\tau\) closer (or greater than) 1 makes result more diverse in the less likely words become more likely.

- Often use a combination of top-\(P\) + Temperature.
  + 1 more repetition penalty - divide by \(1.3\) if 16 word already over

- Demonstration using GPJ3 playground.
  - show code in mingpt generate function
    - temperature
    - torch.multiprocessing | torch.topk
- Demo of GPT-3: basic construction of words
  - shows effect of $I$, $P$, and probabilities.

Now,
- GPT-3 is a 175B parameter model trained on
  $\approx 1$ Trillion words
- It is capable of "zero-shot" learning (a bit of a misnomer)
  meaning that it can do tasks that the model parameters are
  not trained on. How can that be?

  - The class is listening intently.
  - Tokens are intensity!

- hockey. *ad*
  - Click on the

- write a poem/ classify
  - note different settings

- I find zero-shot truly remarkable. How could it work?
  - Is the model writing a computer program to do what is asked?
    - don't think so: struggles to explain, but do have this
  - it is all based on the ability to predict the next word.
  - to do so well one has to know a lot.
  - e.g. how to end: "The car was going faster than a___"
    - need to know a lot

- with very powerful predictor, the language at the
  front becomes an instruction -> the predicted
  words must be consistent with the front instruction
  & so the model does as asked if it can.
- each time inference is run, those words are
  considered: the whole context is the state