ECE 1786 Lecture #6

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 ideation

Bidir vs Unidir
BERT vs. GPT

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 output 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)
- see Jurafsky, section 9.4.3, in context of RNNs.

e.g. START with: The clear river flowed.
Generate: into.
Then next formatted input: The clear river flowed into.
Generate: the
Next input: ~~large~~ The clear river flowed into the
Generate: ...

maybe obvious but be sure you get this - must call the model to inference for each new word.

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

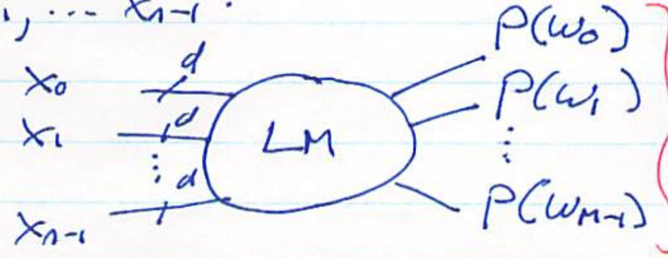
   Generation Parameters (Unknown! i.e. Temperature, Top-P, etc., max tokens generated)

   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 (read

- recall the specific output of the language model, given an input sequence of embeddings:
$x_0, x_1, \dots x_{n-1}$:



where M is the size of the vocabulary.

the probability that each word in vocabulary IS the next word

→ DOESN'T CHOOSE THE WORD.

- So for a given sequence input, which word is selected as output? This is Called Decoding

unfortunate choice.

→ not related to decoder in model

N.B. ⇒ WHAT IS THE BEST ANSWER? → THINK CAREFULLY - WHAT ARE WE LOOKING FOR?

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 sequences of 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 "highly" local optimum

i.e. given input sequence $x_0 \cdots x_{n-1}$ want the generated sequence $y_0 \cdots y_{K-1}$ of K words to be most likely.

i.e. want $P(y_0) \times P(y_1) \times \cdots \times P(y_{K-1})$ to be maximized.

but we don't know $P(y_1)$ when selecting y_0 (or y_j $j>i$ when selecting y_i) UGLY SEARCH PROBLEM.

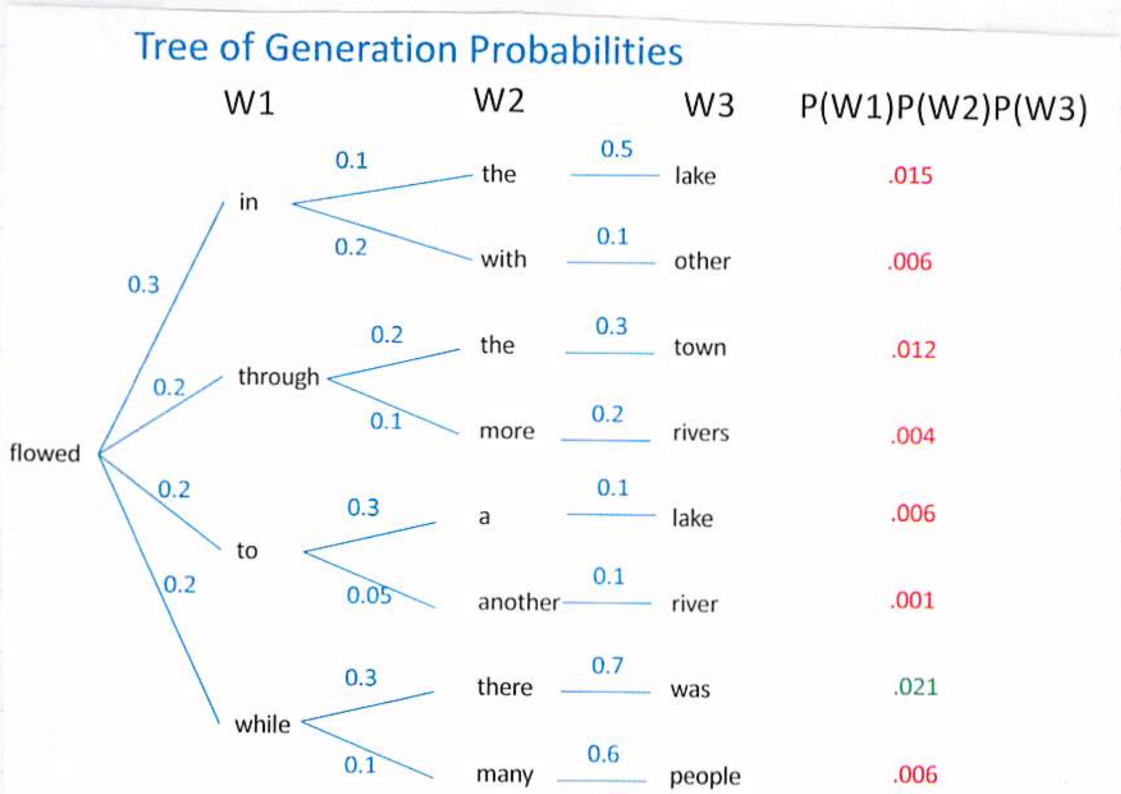
- this is a tough problem because there are M^K possible sequences of K words solutions (e.g. M = 50000, K = 20, 50000^{20} is a big number of large NN model inferences to make!)

↳ i.e. way too many

- per Jurafsky Section 10.5, think of the AUTO REGRESSIVE selection of the output sequence as a tree with probabilities at each layer

Input: The clean river flowed...
Outputs, produced from many invocations of inference using trained model:

give probabilities are made up



the clean river

- so the most probable W1 - "in" doesn't lead to the most probable sequence of W1 W2 W3. .015, .01, .006 vs. .021
- to get the optimal prob sequence is very hard.

Method
(2) Beam Search is a heuristic that prunes the full search tree alot.
- general description: walk ~~through~~ down 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 * V new sequences
  - compute the probability for all by multiply probabilities.
  - keep the K highest.

→ GO DOWN TO NEXT LEVEL.

- each of the K * V a each level costs one inference run through model → expensive! in time.

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

③ ~~Nucleus~~ Sampling.

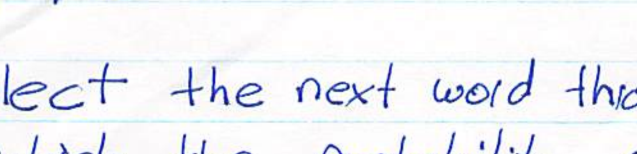
M = vocab size

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

- 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:

  up $P(up) = 0.5$
  down $P(down) = 0.3$
  left $P(left) = 0.2$



then: generate a uniformly distributed random number between 0 and 1.0
- call it R

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

so up will be chosen with probability 0.5; down with prob .3 & left with prob 0.2

- Simplification: Only select from the top K most probable words (top-K sampling)

- most widely used: top-p sampling: only select from ~~those~~ the top words that all together have the sum of probabilities $\geq p$ $(0 < p \leq 1)$ ($p = 1$ means use all; often $p = 0.8$ vastly reduces the number considered)

- also a really unusual adjustment to the sampling method involves an adjustment to the softmax probability calculation @ end of each inference
  - recall $P(w_i)$ = softmax (logits (network output))

l_0, l_1, ..., l_{M-1} let l_i be the logits output of word i.

$$P(w_i) = \frac{\exp\left(\frac{l_i}{t}\right)}{\sum_{\text{all } i=0,M-1} \exp\left(\frac{l_i}{t}\right)}$$

where t is a parameter call Temperature $0 < t$ ~~not~~

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

→ often use a combination of top-p + temperature.
→ +1 more repetition penalty - divide ~~l~~ l_i by $\boxed{1.3}$ if word already used.

- Demonstration using GPT-3 playground.
- show code in mingpt generate function
  - temperature
  - torch.~~mult~~multinomial | torch.topk.

- demo of GPT-3: basic continuation of words
  - show effect of T, P, & probabilities.

Now,
- GPT3 is a 175B parameter model trained on ≈ 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?

IT DOES WHAT IT IS TOLD TO DO.

- e.g The ~~class is listening intently~~ clear river flowed.

- tokens are interesting! → click on them.

- hockey. ad.

- 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; struggled 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 alot.

e.g how to end: "The car was going faster than a ___"
- need to know alot

- 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