## Week 1 – Language Modelling

1. In the context of GPT, what is a Transformer?
   1. A neural network architecture. (Transformers are a specialized type of neural network architecture that GPT uses to model language.)
2. Does the probability provided by a language model depend on a reference corpus?
   1. Yes, there is no single “correct” answer to the probability of specific text because language differs among people, culture, context, and time. (Due to the variety of language, we need to measure the probability of text with respect to a set of reference text.)
3. What type of language model predicts probabilities in the form of P("GPT"|"I","am","learning","about")?
   1. Causal Language Models. (Causal language models predict the final word in a sequence.)
4. What is a key advantage of using a lower 'n' value in an n-gram language model over a higher 'n' value?
   1. Lower 'n' values result in less data sparsity. (This is because there are fewer unique n-grams, which makes it more likely that an n-gram in the test data was also in the training data.)
5. Consider the sentence "The cat is on the mat". If we were to compute the probability of the sentence using a bigram model, how would we compute it?
   1. P("The"|"<BOS>") \* P("cat"|"The") \* P("is"|"cat") \* P("on"|"is") \* P("the"|"on") \* P("mat"|"the") \* P("<EOS>"|"mat")
6. What is NOT a type of n-gram?
   1. Quadrigram. (Quadrigram is not a standard term used in n-gram models. For n equals 4, the term used is four-gram.)
7. What is the effect of increasing 'n' in an n-gram language model?
   1. The model captures more context but requires more computational resources.
8. Why might a tri-gram language model not be suitable for a small corpus of text?
   1. There might not be enough examples of each possible tri-gram to accurately estimate probabilities.
9. Why can measuring the perplexity of a model on its training data be misleading?
   1. It will give an under-estimate of how well the language model generalizes to unseen data. (Since the probabilities were generated based on the training data, you will usually have a lower perplexity on your training data than on unseen data.)
10. What is the best possible value for perplexity?
    1. 1. (A perplexity of one implies that the language model perfectly predicted the sample.)
11. To test how good a language model was trained, your colleague suggests making up a few questions to see how well it answers them. Why is this extrinsic evaluation setup problematic?
    1. A few samples is likely an insufficient amount of data to draw meaningful insights from.
    2. Samples that you make up might not represent a realistic application of your language model.
    3. The samples you make up might be in the training data.
12. Let’s say you compute the perplexity on a held-out test set for two models, a bigram and a trigram language model. The bigram model has a perplexity of 45.12 and the trigram model has a perplexity of 29.64. What can you infer about these two models?
    1. The trigram model is a better fit for the test data than the bigram model.
13. Why is smoothing usually necessary when measuring perplexity of n-gram language models?
    1. The sparsity of language means that without smoothing, many of the probabilities will be zero, resulting in undefined perplexity. (Not only do zero probabilities cause problems when calculating perplexity, but they do not represent the reality of natural language variation.)
14. When a context is encountered that was not seen during training, how do n-gram language models with smoothing estimate token probabilities?
    1. A uniform probability distribution is applied across all tokens. (All tokens are considered equally likely in this case.)
15. How are tokens that were not encountered during training often handled in language models?
    1. They are replaced with a single special “out-of-vocabulary” (OOV) token.
16. Why is perplexity used for intrinsic evaluation of language models?
    1. Perplexity measures how well a language model predicts a sample corpus. (Perplexity measures the probability of a sequence, which directly measures how well the model represents language.)
17. What statement about generating out-of-vocabulary (OOV) tokens is true?
    1. Without smoothing, no generation strategy can generate out-of-vocabulary tokens from an n-gram language model. (Since OOV tokens are given a probability of 0 by n-gram language models.)
18. When considering the trade-offs between different generation strategies, what is a key advantage of using beam search over greedy generation and sampling?
    1. Beam search can generate the more probable sequence.
19. Which method among greedy generation, sampling and beam search is more likely to generate a diverse but less coherent text sequence?
    1. Sampling. (By sampling randomly from the probability distribution, the generated sequences tend to be more diverse and interesting.)
20. In what scenario might greedy generation be a better choice than beam search or sampling for text generation?
    1. When you want to generate more probable sequences
21. What is the purpose of clipping the probability distribution to the “top k” tokens when sampling?
    1. It reduces the chance of generating very low-probability sequences. (Applying a “top k” filter before sampling eliminates very low-probability tokens.)
22. How does the greedy generation method differ from beam search and sampling in terms of text sequence generation?
    1. Greedy generation chooses the word with the highest probability at each step, while beam search maintains a set of most probable sequences, and sampling generates a diverse sequence by introducing randomness.
23. Why might you want to apply smoothing when generating from an n-gram language model?
    1. Smoothing will ensure that every context has some tokens with non-zero probability.
    2. Smoothing will give some probability to sequences not seen during training, improving the diversity of generation.
    3. Smoothing will give some probability to the end-of-sequence token in all contexts.
24. You notice that a greedy generation over a unigram model always repeats the same token indefinitely. What’s happening?
    1. The context added at each step of generation has no effect on the probabilities from a unigram model, so the same token always repeats. (The probabilities in a unigram model are identical, regardless of the context.)