 Name: _____

1 Arrange the following words in the space below so that similar words are closer together: *father*, *mother*, *brother*, *sister*. Then write their x and y coordinates in the table. (tip: use small integers, e.g., −1, 0, 1)

| word | x | y |
|---|---|---|
| father | | |
| mother | | |
| brother | | |
| sister | | |

2 The model reads each context and outputs a vector in the same space as the words above. Where should the model place each of the following contexts so that its dot product is maximized for the correct word? Then compute the dot product of that vector with each of the words (using your coordinates from the table above).

| context | x | y | dot product with *father* | dot product with *mother* | dot product with *brother* | dot product with *sister* |
|---|---|---|---|---|---|---|
| "Martin Jr. was named after his" | | | | | | |
| "At the parent-teacher conference, the boy's" | | | | | | |
| "If Alice is my aunt, then my mother is her" | | | | | | |
| "If Bob is my uncle, then my father is his" | | | | | | |

3 Consider the second row of the context table. Use softmax to compute the *probabilities* of those 4 tokens.

| word | father | mother | brother | sister |
|---|---|---|---|---|
| probability | | | | |

4 Suppose that we're seeing this context during training, and the actual next word is "father".

a Write the expression you'd enter into a calculator for the model's cross-entropy loss for this one prediction:

b What change could we make to the $x$ and $y$ columns in the Questions 1 and 2 tables to reduce the model's loss?

Write Python-like pseudocode for an **algorithm** to complete a sentence.

- **Input**: a sentence `prefix` (string)
- **Output**: a completion of that sentence (string)

You can assume that sentences end with a special end-of-sentence token, which the model will predict when it thinks the sentence is complete. Your algorithm should use the following resources:

- `tokenize(text: str) -> tensor[seq_len]`: returns a tensor of token ids
- `model(token_ids: tensor[seq_len]) -> tensor[seq_len, vocab_size]`: returns a tensor of next-token logits (assume no batching, so it's just two axes)
- `decode(token_ids: tensor[seq_len]) -> str`: returns the text that would `tokenize()` into those `token_ids`.
- `softmax(array: tensor[vocab_size]) -> tensor[vocab_size]`: returns `exp(array) / sum(exp(array))` (the softmax function)
- `sample(probs: tensor[vocab_size]) -> int`: returns a random integer in the range `[0, len(probs)-1]` proportional to the corresponding probabilities (`probs` must be a valid probability distribution)
- global constant `END_OF_SENTENCE_TOKEN_ID`