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 simple numbers between −1 and 1)

word	x	y
father		
mother		
brother		
sister		

2 Now place the following contexts in the same space and give their coordinates. Then compute the dot product with each of the words (using the coordinates you filled in in 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 context "At the parent-teacher conference, the boy's _____". Based on the table above, compute next-token *probabilities* for one of the phrases, assuming that we're only considering the words in the table.

word	father	mother	brother	sister
probability				

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

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

b What change could we make to the x and y columns in each table above to reduce the model's loss?

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

- **Input**: a sentence `prefix` (string), a `temperature` parameter (float)
- **Output**: a completion of that sentence (string)

Your algorithm should use the following resources:

- `tokenize(text)`: returns a tensor of token ids
- `model(token_ids)`: returns a tensor of next-token logits (assume no batching, so it's just two axes)
- `decode(token_ids)`: returns the text that would `tokenize()` into those `token_ids`.
- `sample(probs)`: 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`

```
def complete_sentence(prefix, temperature):
    token_ids = tokenize(prefix) # list of token ids
    while True:
        logits = model(token_ids) # next-token logits for every token (len(toks), n_vocab)
        next_token_logits = logits[-1] # (n_vocab,)
        probs = softmax(next_token_logits / temperature)
        next_token_id = sample(probs) # next_token_id is a number [0, n_vocab - 1]
        if next_token_id == END_OF_SENTENCE_TOKEN_ID:
            break
        token_ids.append(next_token_id)
    return decode(token_ids)
```