1 Definitions

1.1 Word Embeddings

Some terms that may be helpful to know while going through this activity

1. **Corpus**: A corpus refers to a large and structured set of texts in a specific language or domain, typically used for linguistic analysis, training of language models, and other NLP tasks.
2. **Token**: In NLP, a token represents a single unit of a sequence, typically a word or a punctuation mark, obtained by splitting the input text based on predefined rules.
   For example: In the sentence "The cat sat on the mat.", the tokens are: "The", "cat", "sat", "on", "the", "mat", and ".".
3. **Word Embedding**: A word embedding is a technique used to represent words from a vocabulary in a lower-dimensional vector space

2 Word Embeddings

We will be using the notebook found in the google drive folders for your recitation.

This version has two $vocab\_size \times 2$ matrices of parameters, V and U, used to encode and decode tokens, respectively. But rather than trying to decode the same word as the input like an autoencoder, we are trying to predict the next word.

Another way to think about these two parameter matrices is V maps *prev* token indices into a 2-dimensional space and U maps *next* token indices into the same 2-dimensional space. In that 2-dimensional space, we can match previous tokens to their corresponding next tokens by using the cosine similarity metric, $\mathbf{v}^\top \mathbf{u}$.

This notebook loads the lines in Green Eggs and Ham and trains our word embeddings model for the corpus from that story. Our goal is to use this as a language model to generate the likely next token, given a specific previous token.

1. What technique can we use to determine the closeness of two tokens/words? How can we use it?

   We can use cosine similarity to determine the similarity between two vectors.

   Cosine similarity measures the cosine of the angle between them. It ranges from -1 to 1, where 1 indicates perfect similarity (the vectors point in the same direction), 0 indicates no similarity (the vectors are orthogonal), and -1 indicates perfect dissimilarity (the vectors point in opposite directions).

   We can simplify this calculation to determine the dot product between the two vectors. That is, the vectors are similar if the dot product is large.

2. After implementing the similarity function and completing the objective implementation, run the training loop. What final loss do you obtain for the entire corpus at the start and after 100 epochs?

   Start: 4.6891
   100 epochs: 2.9759

3. Run the notebook and observe the embedded vectors change during the training loop for 100 epochs.

   The notebook then loads the pre-trained word embeddings after 27,000 epochs. Compare the visualization of the purple U next word embeddings after 100 epochs to after 27,000 epochs. What organization do you observe in the trained embedded space?

   More spread out, equal magnitude of vectors.

   Use this simple Word2vec example as a language model to generate new text!

   (a) Current context, e.g., "i will not"
   (b) Previous word: "not" (last word of current context)
   (c) Previous word index from `token_to_vocab_index`
   (d) Use our model to get $\hat{y}$, i.e., the next-word probabilities for each possible vocab word
   (e) Select the most probable next token (`np.argmax`)

   Give an example of some of the text that you generate:

   Run the solution notebook cell a few times to see generated text.

   As a bonus, sample the next token using the `yhat` probabilities (`np.choice`, using the optional argument `p=yhat`). Give an example of some of the text that you generate:

   Switch the sample flag to True to see this generated text.