Group members: Daniel Yoon, Nicole Reardon, Lukas Fullner, Andrew Tseng

Everybody put in the same amount of effort in performing this assignment. All the problems are reflective of the group's discussions during the many meetings performed.

Problem 1

The function load_corpus opens the 'sample_corpus.txt' file then reads in every line, replacing the newline characters with spaces to make it one long string. The segment_and_tokenize function processes a given text corpus into a list of tokenized sentences. It begins by loading the SpaCy English language model to analyze the text which identifies linguistic features like sentences and tokens. It then segments the text into sentences and tokenizes each sentence into individual words, filtering out sentences with fewer than two tokens to exclude short, uninformative fragments. The function then uses remove_infrequent_words to replace words that appear less than twice in the corpus with the <UNKNOWN> token. Then, the function adds <START> and <END> tokens to signal the beginning and ending of each sentence. Finally, it returns the list where each sentence is represented as a list of tokens.

Problem 2

The make_word_to_ix function maps unique words to unique indices from a list of tokenized sentences. It initializes an empty dictionary and a counter (starting at 0) to track the number of unique words. For each word in every sentence, it checks if the word is already in the dictionary. If not, it assigns the current value of the counter as the word's index in word_to_ix and increments the counter. This ensures that each unique word in the corpus receives a distinct integer. The function then returns the dictionary mapping words to indices.

Problem 3

The vectorize_sents function converts tokenized sentences into one-hot vector representations based on the mappings created by make_word_to_ix. It starts by initializing an empty list to store the one-hot encoded representations of each sentence. For every sentence, it calls sent_to_onehot_vecs, which converts the words in the sentence into one-hot vectors using their indices from word_to_ix. A one-hot vector has a length equal to the vocabulary size of 0s, then sets at the index corresponding to the word with a value of 1. The function appends the resulting list of vectors for each sentence to one_hot_vecs and returns it.

Problem 4

The embed_word function generates a word embedding for a given word using the embedding weight matrix W_e and a one-hot vector. Each current_word is represented as a one-hot vector, which has all zeros except for a 1 at the position corresponding to the word's index in the

vocabulary. The function performs matrix multiplication between W_e and the one-hot vector, effectively extracting the column of W_e corresponding to the word's index. This results in a dense embedding vector of size embedding_dim that captures semantic similarities between words.

Problem 5

The elman_unit function computes the hidden state for the current step. The function performs matrix multiplication between word_embedding, W_x*x_t, and the weight matrix W_x, between h_previous and the recurrent weight matrix W_h, adds a bias term b, and applies a sigmoid activation function to the result. This operation updates the hidden state to incorporate both the current word and past context.σ = (W_x*x_t + W_h*h_previous + b).

Problem 6

The single_layer_perceptron function computes the predicted probability distribution over the vocabulary for the next word. It takes h_current as the input and applies a linear transformation using the output weight matrix W_p. The result is passed through a softmax function, which normalizes the output into a probability distribution. This represents the model's prediction for the likelihood of each word in the vocabulary appearing next. softmax = $e^{WpHcurrent*i}/\Sigma_k\ e^{WpHcurrent*k}$ with i representing the current word, and k representing the entire set of words.

Problem 7

The Elman network enforces causality by processing words sequentially, one at a time, in the order they appear in the sentence. At each step, h_current is computed only based on the current input word and h_previous, in which future words are not used in this computation. This ensures that the model does not have access to information from future steps while predicting the next word, maintaining causality in the language modeling process.

Problem 8

The first contributes directly to h_current when current_word is the second word of the text because in the elman_unit function, the sigmoid is taken on two matrix multiplications that represent encoded information about the current word and encoded information about the previous word(s) (which in this case is only the first word). The first word has a smaller influence on the third word as W_h □* h_previous now encompasses information about the first two words and the hidden state updates prioritize more recent inputs. It’s analogous to how grandparents’ genes can influence their grandchildren’s genes.

Problem 9

When the train function is executed, the loss decreases over epochs as the model learns to predict the next word more accurately. This decrease in loss indicates that the probability distribution generated by the model for the next word aligns better with the observed word distributions in the training data. A consistently decreasing loss implies that the RNN is successfully learning patterns and relationships in the data, becoming more confident and precise in its predictions.

Problem 10

Some hyperparameter modifications that resulted in good model performance on the training set included setting num_epochs to a higher value (150 - 200), hidden_state to 40, leaving embedding_dim to 20, and increasing the learning rate to 0.01. From the default parameters of 100/20/20/0.001, this resulted in a loss of 5.020697 during one run, but we were able to achieve a loss of 4.7973146 with these changes: 150/40/20/0.01.

Given that the hidden_state_dim and embedding_dim are the same values, whether they be high or low, if the learning rate were low (such as 0.001) then the initial loss would be great (>7). We further noticed that the greatest difference in loss would occur between the first and second epochs, where loss would decrease the most.

Problem 11

The Elman network would likely not be able to model the dependency between the 3rd and 950th word in the text due to the recurrent nature of the network, where information from earlier words is passed through a chain of hidden states. Over many timesteps, the gradients used to update the weights during training diminish causing the network to "forget" distant inputs.

Problem 12 on following pages

Problem 12

12.1)

$h_1 = \sigma(W_x \cdot x_1 + W_h \cdot h_0)$

$= \sigma\left(\begin{bmatrix}0.5\\0.3\\0.2\end{bmatrix} \cdot \begin{bmatrix}1\\0\\0\end{bmatrix} + 0.5 \cdot 0\right)$

$= \sigma(0.5 + 0) = \sigma(0.5)$

≈ 0.622

$h_2 = \sigma(W_x \cdot x_2 + W_h \cdot h_1)$

$= \sigma\left(\begin{bmatrix}0.5\\0.3\\0.2\end{bmatrix} \cdot \begin{bmatrix}0\\1\\0\end{bmatrix} + 0.5 \cdot 0.622\right)$

$= \sigma(0.3 + 0.311) = \sigma(0.611)$

≈ 0.648

$y_1 = \text{softmax}(W_p \cdot h_1)$

$= \text{softmax}(0.622\,[0.1 \;\; 0.2 \;\; 0.1])$

$= \text{softmax}([0.0622 \;\; 0.1244 \;\; 0.0622])$

$\approx \begin{bmatrix}0.326\\0.347\\0.326\end{bmatrix}$

$y_2 = \text{softmax}(W_p \cdot h_2)$

$= \text{softmax}(0.648\,[0.1 \;\; 0.2 \;\; 0.1])$

$= \text{softmax}([0.0648 \;\; 0.1296 \;\; 0.0648])$

$\approx \begin{bmatrix}0.326\\0.348\\0.326\end{bmatrix}$

$L_1 = -\log_e(y_1[\text{target 1}])$

$= -\log_e(0.347)$

≈ 1.058

$L_2 = -\log(y_2[\text{target 2}])$

$= -\log(0.326)$

≈ 1.121

$L = L_1 + L_2 = 1.058 + 1.121 = 2.179$

A functioning link to the PDF for problems 12.2 and 12.3 can be found here

Problem 12.2

Given the following:

$$L = L_1 + L_2$$

We first compute $\frac{\partial L_1}{\partial W_h}$. Using the chain rule, we arrive at the conclusion:

$$\frac{\partial L_1}{\partial W_h} = \frac{\partial h_1}{\partial W_h} \cdot \frac{\partial L_1}{\partial h_1}$$

The derivative $\frac{\partial h_1}{\partial W_h}$, given that $h_1 = \sigma(W_t x_1 + W_h h_0)$, is evaluated as:

$$\frac{\partial h_1}{\partial W_h} = h_0 \cdot \sigma'(W_t x_1 + W_h h_0)$$

It is given that $h_0 = 0$, meaning:

$$\frac{\partial h_1}{\partial W_h} = 0$$

Consequently, this leads us to the conclusion that, for $\frac{\partial L_1}{\partial W_h}$,

$$\frac{\partial L_1}{\partial W_h} = 0 \cdot \frac{\partial L_1}{\partial h_1} = 0$$

We now compute $\frac{\partial L_2}{\partial W_h}$, which, through the chain rule, comes to the conclusion:

$$\frac{\partial L_2}{\partial W_h} = \frac{\partial h_2}{\partial W_h} \cdot \frac{\partial L_2}{\partial h_2}$$

In the same way we compute $\frac{\partial h_1}{\partial W_h}$, we compute $\frac{\partial h_2}{\partial W_h}$

$$\frac{\partial h_2}{\partial W_h} = h_2 \cdot \sigma'(W_t x_2 + W_h h_1)$$

Given that we are asked to take the derivative with respect to the target index, which in the case of $t = 2$, is 2, we take the partial derivative of following equation, with respect to h_2:

$$\mathrm{L}_2 = -\log(\text{softmax}(W_p h_2)[2])$$

The derivative of L_2 itself is given as

$$\frac{\partial L_2}{\partial h_2} = -\frac{1}{y_2[2]} \cdot \frac{\partial y_2}{\partial h_2}$$

The proof of the derivative of softmax is out of the scope of this course, but given that t and the target_index $= 2$, the derivative of $y_2 = \text{softmax}(W_p h_2)$ is given as:

$$\frac{\partial y_2}{\partial h_2} = y_2[2] \cdot (1 - y_2[2]) \cdot W_p[2]$$

The chain rule has been performed on the input of softmax, $W_p h_2$, but has been omitted for the sake of time.
Thus it now holds that

$$\frac{\partial L_2}{\partial h_2} = -\frac{y_2[2] \cdot (1 - y_2[2]) \cdot W_p[2]}{y_2[2]} = -(1 - y_2[2]) \cdot W_p[2]$$

Given that $\frac{\partial L_2}{\partial W_h} = \frac{\partial h_2}{\partial W_h} \cdot \frac{\partial L_2}{\partial h_2}$, we now come to the conclusion

$$\frac{\partial L_2}{\partial W_h} = h_2 \cdot \sigma'(W_t x_2 + W_h h_1) \cdot -(1 - y_2[2]) \cdot W_p[2]$$

Since we have shown earlier that $\frac{\partial L_2}{\partial W_h} = 0$, it holds that:

$$\frac{\partial L}{\partial W_h} = 0 - h_2 \cdot \sigma'(W_t x_2 + W_h h_1) \cdot (1 - y_2[2]) \cdot W_p[2]$$

Problem 12.3

The calculations for the variables are considered trivial, and thus omitted from this document.

$$h_1 = 0.622$$

$$h_2 = 0.648$$

$$\sigma'(W_t x_2 + W_h h_1) = \sigma'(0.611) = 0.228$$

$$y_2[2] = 0.326$$

$$W_p[2] = 0.1$$

Using the equation for $\frac{\partial L}{\partial W_h}$ derived from Problem 12.2, we get the numerical value:

$$\frac{\partial L}{\partial W_h} = 0 - 0.648 \cdot 0.228 \cdot (1 - 0.326) \cdot 0.1 = -0.009957$$